Instituto Politécnico de Coimbra

Instituto Superior de Engenharia de Coimbra

Departamento de Engenharia Informática e de Sistemas

Machine-to-Machine Emergency System for Urban Safety

André Filipe Gomes Duarte

Mestrado em Engenharia Informática

Coimbra, dezembro, 2015

Instituto Politécnico de Coimbra

Instituto Superior de Engenharia de Coimbra

Departamento de Engenharia Informática e de Sistemas

Mestrado em Engenharia Informática

Estágio

Relatório Final

Machine-to-Machine Emergency System for Urban Safety

André Filipe Gomes Duarte

21200791

Orientador: Professor Doutor Jorge Bernardino

Orientador da Empresa: Engenheiro Carlos Oliveira

Ubiwhere

Coimbra, dezembro, 2015

Acknowledgments

I would like to express the deepest appreciation to my supervisor at Ubiwhere, Carlos Oliveira, for his availability, motivation and knowledge shared during the internship.

To Professor Jorge Bernardino for his accessibility and enthusiasm not only during the internship, but also during the entire degree.

To Ricardo Vitorino for his availability, knowledge and sharing useful inputs while the writing of the report.

To the remainder of my professors at ISEC which shared their knowledge and experience.

To the Ubiwhere team, which provided an excellent working environment where it was possible to share knowledge and learn many new things.

To my friends and colleagues not only for their support but also for sharing experiences and knowledge.

To my family and girlfriend for supporting me in another stage of my life, providing strength and willpower to overcome every challenge.

Resumo

Atualmente a maioria das pessoas vive em áreas urbanas. Com o crescimento das populações a exigência sobre os ecossistemas da cidade aumenta, afetando diretamente as entidades responsáveis pelo seu controlo. Desafios como este, fazem com que os responsáveis das cidades adotem maneiras de se interligar com o meio envolvente, tornando-os mais preparados e conscientes para a tomada de decisão. As decisões que tomam não só afetam diretamente a cidade a curto prazo, mas são também um recurso para melhorar o processo de tomada de decisão.

Este trabalho teve como objetivo desenvolver um sistema que pode agir como supervisor de emergência e de segurança numa cidade, gerando alertas em tempo real, que fornecem às entidades responsáveis novas competências para garantir a segurança. Este sistema é capaz de monitorizar os dados de sensores e fornecer conhecimento útil a partir deles.

Este trabalho apresenta uma arquitetura para a recolha de dados na Internet das Coisas (IoT), proporcionando ainda a análise das ferramentas utilizadas e as escolhas feitas sobre o sistema implementado. Além disso, fornece os elementos necessários para que novos colaboradores possam vir a participar no projeto, uma vez que descreve todas as técnicas, linguagens, estratégias e paradigmas de programação utilizados.

Finalmente, descreve o protótipo que recebe os dados e os processa para gerar alertas com o objetivo de avisar equipas de emergência, descrevendo ainda a futura implementação de um módulo de previsão que pode agir como uma ferramenta útil para melhorar a gestão de equipas de emergência.

A realização do estágio permitiu a aprendizagem de novos conceitos e técnicas, bem como o desenvolvimento daqueles que já estavam familiarizados. No que diz respeito à empresa, o sistema desenvolvido irá integrar a plataforma Citibrain funcionando como um ponto central, no qual, cada aplicação (por exemplo, gestão da água, gestão de resíduos) se poderá subscrever para receber alertas.

Abstract

Nowadays most people live in urban areas. As populations grow, demand on the city ecosystem increases, directly affecting the entities responsible for the city control. Challenges like this make leaders adopt ways to engage with the surroundings of their city, making them more prepared and aware. The decisions they make not only directly affect the city in short term, but are also a means to improve the decision making process.

This work aimed to develop a system which can act as an emergency and security supervisor in a city, generating alerts to empower entities responsible for disaster management. The system is capable of monitoring data from sensors and provide useful knowledge from it.

This work presents an architecture for the collection of data in the Internet of Things (IoT). It delivers the analysis of the used tools and the choices made regarding the implemented system. Also, it provides the necessary inputs for developers to participate in the project, since it describes all the techniques, languages, strategies and programming paradigms used.

Finally, it describes the prototype that receives data and processes it to generate alerts with the purpose of warning emergency response teams and the future implementation of a prediction module that can act as a useful tool to better manage the emergency personnel.

The completion of the internship allowed the learning of new concepts and techniques, as well as the development of those that were already familiar. With regard to the company, the developed system will integrate the company's Citibrain platform and will act as a central point, in which, every application (e.g. water management, waste management) can be subscribed to receive alerts.

Keywords:

Disaster Management

Emergency Systems

Smart Cities

Urban Safety

Index

Index of Figures

1. INTRODUCTION

Nowadays most people live in urban areas. The growth in population directly affects the city ecosystem and the entities responsible for the city control. City leaders must be aware of these changes and approve ways to engage with the surroundings of their city, enhancing their awareness and preparedness. The decisions they make not only directly affect the city in a short term, but are also a means to improve the decision making process. The growth in the number of human beings in urban areas comes with a significant increase in data. This data comes from sensor networks scattered around the city or from the sensors in a smartphone. As data production and availability increased, so did the need to integrate it and provide value added services to citizens.

As smart cities mature, legacy systems already in place are trying to evolve to become smarter, although these systems have many specific requirements that need to be attended. An architecture which is scalable, adaptable and interoperable for the Internet of Things (IoT) is necessary, therefore existing architectures will be analysed as well as the algorithms that make them work.

In this work we propose an architecture that is scalable, adaptable and interoperable for the Internet of Things (IoT) environment. It will be used as a basis to develop a system that will integrate the company's Citibrain Platform. Using sensors this system will monitor parameters such as temperature, humidity, atmospheric pressure, ultraviolet radiation, fire detection and flooding in various points of a city. With this data comes the ability of providing useful knowledge that will lead to the alerting of the safety experts. These alerts are generated by the system when it analyses the information, which can provide indicator trends or make predictions for future events (e.g. fires, floods). The alerts intend to provide the ability for authorities to act as soon as possible upon an imminent danger.

This internship aims to develop a system that monitors city parameters, such as temperature, humidity, ozone, amongst others. The main idea behind the monitoring of these parameters is to understand whether the security thresholds are in risk of being overtaken and raise alerts that will notify the security personnel of a forthcoming danger. In short, one of the main goals is to create an intelligent support system, which monitors city parameters, generates alerts and facilitates the decision-making process.

With the purpose of improving the citizen's quality of life and quickly and efficiently make informed decisions, authorities try to monitor all the information of city systems. Smart cities provide the integration of all systems in the city via a centralized command centre, which provides a holistic view of it. With the intent to suit the needs of specific systems the focus of this work is to gather viable information that leads to the analysis and presentation of solutions to address their current shortcomings. A system, based on the the developed architecture, will be implemented and integrated with the company's Citibrain platform. The applications that are already deployed in the platform will provide information in real time, therefore the role of the system is to analyse this raw data, fire alerts and make predictions in order to mitigate future dangers. From a practical standpoint, this work intends to develop a Machine to Machine (M2M) prototype to act as an emergency and security supervisor.

1.1 Main Contributions

The main contributions of this work are:

- An Application Program Interface (API), which intends to gather data from sensors scattered around the city;
- An engine capable of receiving, processing and dispatching the alerts to emergency personnel;
- A predictive analytics module which provides the capability of estimating future events;
- The documentation of the best possible way to develop a smart system that receives streams of data from many sources and provides knowledge from it;
- The creation of a prototype, based on the lambda architecture (Lambda Architecture, 2014), which provides tools to work both with streams and batches of data, empowering cities with the necessary knowledge to avoid some types of disasters;
- A generic IoT architecture for anyone to use or improve;
- A prototype that analyses data and provides indicators and alerts to experts.

1.2 Ubiwhere

Ubiwhere is a software company, created in 2007, based in Aveiro and with offices in Coimbra and São João da Madeira, which specializes in research and innovation, idea to product and user-centred solutions. There are many brands under Ubiwhere's name such as Citibrain, rprobe and Playnify.

This project is integrated in Citibrain, which is "a consortium which specialises in smart solutions for today's cities. Headquartered in Aveiro, Portugal, the consortium's main purpose is to create desirable and liveable places, bringing together cities and citizens to improve metropolitan life. Creativity, knowledge and innovation are at the core of Citibrain's strategy" (Citibrain, 2015).

1.3 Structure of the Report

The rest of this report is divided in the following chapters:

2 State of The Art – This chapter will cover the most important concepts related to smart cities and smart emergency systems;

3 Proposed Architecture – This chapter describes the proposed architecture for an Internet of Things system;

4 System Database Evaluation – This chapter explains how the database in the system is structured and handles the data;

5 Alerts Module Implementation – This chapter provides a technical approach to the main component of the system, which is the module that generates all the alerts. A broad perspective of the system is given to provide a better understanding on how the modules work;

6 Predictive Analytics Module – This chapter describes the predictive analytics module, which aims to make predictions for future disasters (e.g. fire, floods);

7 Case Study: Urban Safety system within Oporto's Citibrain node – This chapter shows a use case of the developed prototype with the data available from the Oporto's network of environmental sensors;

8 Conclusions and Future Work – This chapter addresses the general conclusions and developed work, also including a note on work to be done in the future;

Annex A – Internship proposal – This annex refers to the internship document proposed by Ubiwhere;

Annex B – Smart Cities: An architectural approach – This is the paper published and presented at the International Conference on Enterprise Information Systems (ICEIS), which was held at Barcelona from the 27th to the 30th of April 2015.

Annex C – Cassandra for Internet of Things: an experimental evaluation – This paper was submitted to the International Conference on Internet of Things and Big Data (IoTBD), which will be held at Rome from the 23th to the 25thof April 2016.

2. STATE OF THE ART

The area surrounding Internet of Things (IoT), Smart Cities and Smart Emergency Systems is vast and very much in its early stages. In this chapter we will attempt to analyse the most relevant parts of this ecosystem. The addressed problem has been partially developed in the past years with other studies and projects. To understand the basis of the developed work the necessary background shall be provided. It is important to acknowledge that the documented analysis will be high-level, although it covers as much information as possible.

2.1 Smart City

Smart cities are usually defined as modern cities with smooth information processes, streamlined mechanisms for creativity and innovativeness and sustainable solutions promoted through service platforms.

A smart city depends on the provision of information, communication technologies and services to the population via web based services (Alazawi et al., 2014). However, this formulation of Smart City can be misleading. In order to be smart, a city does not need state of the art technology, but interoperability between various key aspects of the city, such as governance, finance, transportation and many others. The kind of changes that smart cities will bring to the current world are many times said to be as similar to those seen in the industrial revolution. The motivation behind the concept is the ability to improve the city's ecosystem while focusing on people and allowing technology to work for them and not with them, thus resulting in a greater vision of society.

There is a wide variety of city concepts that have built a new horizon for cities in their challenging tasks in an increasingly cost-consciousness, competitive and environmentally oriented setting. Irrespective of whether the concept is smart city, intelligent city, sustainable city, knowledge city, creative city, innovative city, ubiquitous city, digital city or city 2.0 (e.g. (Komninos, 2002; Aurigi, 2005; Carrillo, 2006; Hollands, 2008)) they all define a standard of a modern city with smooth information processes, facilitation mechanisms for creativity and innovativeness, plus smart and sustainable service solutions and platforms (Anttiroiko et al., 2014). However, there is still a general absence of joint planning by city governments with utility providers (e.g. water, in respect of environmental sustainability) and other public services (e.g. health care). Cultural barriers

include commercial confidentiality, whereas social media user groups work with open data systems, causing problems for joint working of cities with the private sector. This may create problems for collaborative ventures between city governments and businesses, and even with other public sector agencies, as well as with voluntary and community organisations.

The smart city concept can vary from the technologies and infrastructures of a city to an indicator that measures the education level of its inhabitants (Vakali et al., 2014). Furthermore this work intends to analyse the SEN2SOC (SENsor to SOCial) experiment for its impact in the current context of this topic. The SEN2SOC experiment promotes interactions between sensors and social networks to enhance the quality of data in the city of Santander. The concept of smart city is also referred and conceptualized in (Chourabi et al., 2012). The work enlists some success factors for smart cities, which are: (1) management and organization; (2) technology; (3) governance; (4) policy context; (5) people and communities; (6) economy; (7) built infrastructure; (8) natural environment. These factors provide a solid basis for the comparison of how cities are defining their smart initiatives. Also they represent the key areas for the success of every smart city.

A theoretical definition of Smart City is yet to be found, although cities are developing and shaping for a not so distant future (Piro et al., 2014). Furthermore this work enlists some of the current definitions for the concept, that there is yet to be completely defined. In (Piro et al., 2014) it is also referred the necessity of Information and Communication Technology (ICT) services, with the intent to integrate them in a generic scenario of a smart city. The approach is from a service point of view, which means that it emphasises the role of the services in the city. It is also important to refer that real world cases are shown to prove the importance of the topic.

2.2 Smart Cities Examples

There are many examples of smart cities such as Amsterdam (Amsterdam Smart City, 2014), Santander (Santander Facility, 2014) and Barcelona (Barcelona Open Cities Challenge, 2014). These cities, due to constant innovation projects and investments, have a tendency to be pioneers in the adoption of new standards for smart cities. These cities use smart applications to facilitate the decision making process of their leaders.

In Finland, the city of Helsinki is running a cooperation cluster called Forum Virium Helsinki (Forum Virium Helsinki, 2014) to provide a platform to develop ICT-based

services in cooperation with enterprises, public authorities and citizens as end-users. Although the work presents five project areas, the most relevant for our work is a smart city initiative focusing on the development of mobile phone services to facilitate urban travelling and living. It also opens up public data so that companies and citizens can create new services by combining and processing the data in innovative ways. This resembles the LivingLab movement that has spread across Europe in the 2000s (European Network of Living Labs, 2014).

In the city of Santander there are sensors to monitor the environment, parking areas, parks, gardens and irrigation systems. These sensors are scattered around the city in order to produce alerts that will notify end users with the status of the key aspects of the city.

The data is captured by an IoT node that monitors indicators such as temperature, noise level or luminosity. This data then flows through repeaters positioned in higher grounds, which send it to gateways. Lastly, this data is stored in a database or sent to other machines where it is needed.

Regarding the environmental scenario, from a user's point of view, the available indicators are the temperature, Carbon dioxide (CO_2) level, luminosity and noise, which allow them to receive useful inputs for their wellbeing throughout the day. These indicators are integrated in the environmental monitoring system, which intends to monitor the status of the city.

The environmental monitoring system is important because it shows how sensors interact with the server and how the server communicates back to the sensors and other subscribers that need this type of information.

The Santander City provides another system, named "Participatory Sensing" (Description of implemented IoT services, 2014). This system allows users to actively participate in the city ecosystem (e.g. by publishing an event in the platform). The information is then sent to the SmartSantander platform. Additionally, users become subscribers of the city systems and are able to receive updates of the current status of the road they have to cross to reach their destination. This type of instant real-time information directly affects the city from a user's point of view due to its constant availability and usefulness. An application is available for smartphones and users without smartphone can interact with the platform via SMS.

Figure 1 illustrates the concept of participatory sensing from a user's point of view, which helps to understand how a typical user interacts with this kind of technologies and also

how they provide useful inputs to understand the type of data a user needs during application usage. It is possible to visualise that a user can, in this case, publish events, search for events, visualise historical data, subscribe and unsubscribe to events and receive notifications.

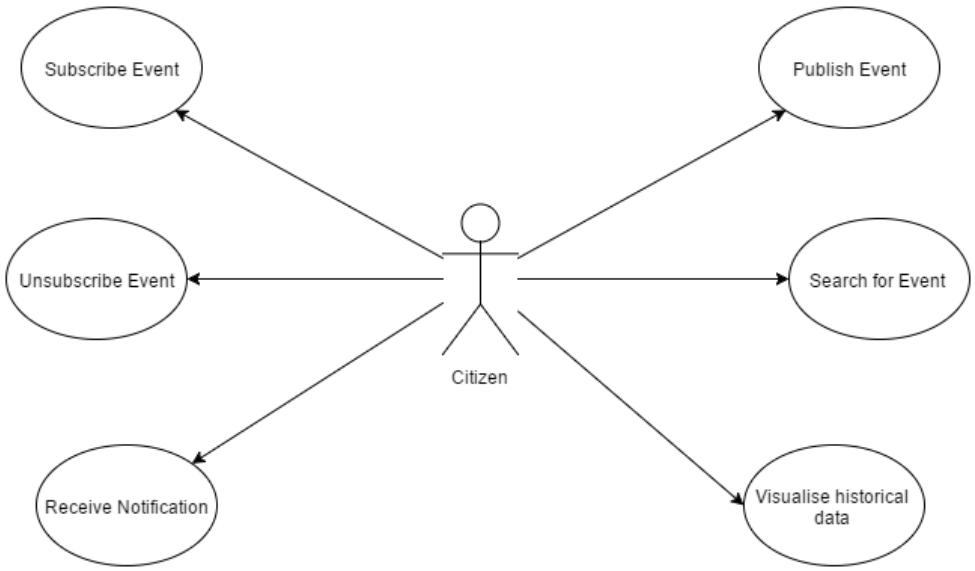

Figure 1 - Participatory Sensing - Use Case Diagram (adapted from (Description of implemented IoT services, 2014))

The components of the participatory sensing system are: a mobile client for end users to utilise; a server, capable of iterating through data and providing links between the apps and the SmartSantander platform also known as “Pace of The City Server”; and a module that allows devices to register onto the platform. Also, there is a system called “Universal Alert System” (UAS) system, which aims to fire user notifications.

Additionally, Santander provides other interesting case studies, which are “Precision Irrigation” and “Smart Metering”.

Precision irrigation is a service that intends to provide a useful way of monitoring plants’ necessities and guarantee that they are fulfilled. Rather than being applied to a whole park, this system is applied to sections or individual plants. Also, the system not only focuses on water management but also on other plant needs, considering their species and growth patterns to minimize the staff effort. This system showed the necessity of designing a solution which accepts communications with REST and WebSockets, due to being two of the most important protocols and well-accepted service patterns when dealing with data in smart cities.

Smart Metering system aims to provide IoT based solutions to monitor energy usage in offices. To address this problem new components have been added to the architecture to generate, collect and store the data and information. In addition to these, intelligent components have also been created in order to provide useful information in a user-friendly way. These components provide data analysis in real-time and consequent knowledge extraction to identify energy failures and generate reports on energy consumption.

2.3 Internet of Things (IoT)

According to (Friess and Vermesan, 2013) the Internet of Things (IoT) "is a concept and a paradigm that considers pervasive presence in the environment of a variety of things/objects that through wireless and wired connections and unique addressing schemes are able to interact with each other and cooperate with other things/objects to create new applications/services and reach common goals. In this context the research and development challenges to create a smart world are enormous. A world where the real, digital and the virtual are converging to create smart environments that make energy, transport, cities and many other areas more intelligent."

Internet of Things is a concept reflecting a connected set of anyone, anything, anytime, anyplace, any service, and any network (Islam et al., 2015). The IoT is a megatrend in next-generation technologies that can impact the whole business spectrum and can be thought of as the interconnection of uniquely identifiable smart objects and devices within today's Internet infrastructure with extended benefits. Benefits typically include the advanced connectivity of these devices, systems, and services that goes beyond machine-to-machine (M2M) scenarios (Höller et al., 2014). Therefore, introducing automation is conceivable in nearly every field. The IoT provides appropriate solutions for a wide range of applications such as smart cities, traffic congestion, waste management, structural health, security, emergency services, logistics, retails, industrial control, and health care.

The Internet of Things comprises the full ecosystem of data in smart cities, which in other words means that IoT generates massive amounts of data that need to be processed by algorithms and tools with the intent to be useful for a city (Jara et al., 2014). This will also provide new ways to interact with intelligent devices and create homogeneous platforms that include both machines and humans working together. Still according to (Jara et al., 2014) this new paradigm will shape the world and create a new concept of

Internet and how people interact with it, due to the constant interconnectivity between people and the world. It will also provide the necessary resources for the creation of new applications and data driven platforms that will, hopefully, improve the citizen's quality of life. This new way of reinventing the Internet will not only provide endless possibilities to improve the overall interaction between humans and machines but also create new challenges, which need to be tackled, to cities themselves. Furthermore, the work aims to develop data-driven models based on human actions to act as a proof of concept for Smart Cities. Additionally, the work concludes that the devices in the Internet of Things are able to gather data and provide knowledge and that a new age of interaction is about to appear, due to the increasing demand for smart applications.

2.4 Emergency Management

Emergency Management is the process that continuously prepares for a disaster even before it happens (Feng and Lee, 2010), intending to protect people from natural or man-made disasters. It is expected that it can integrate many emergency sources to provide the best possible outcome for the situation. In (Feng and Lee, 2010) the authors conclude that emergency management is of extreme importance in the nowadays world.

In (Benkhelifa et al., 2014) the authors listed the current disaster management projects. The purpose of this work is to summarize existing projects regarding this matter. This work is relevant due to its diversity and detail while presenting the projects, which is extremely important to have a baseline of what has already been studied and how it can, if possible, be improved. It is important to state that the focus of this work is wireless sensor networks, being the most relevant outputs the knowledge and awareness of the projects in this area. Also, the work delivers a wider perspective about the topic and led to discoveries regarding the State of the Art projects, which by itself ignited the discovery of solutions and use cases for each problem.

One of the major problems encountered when dealing with large amounts of data is the system's vertical scalability. It is also important to understand how similar systems operate when larger amounts of data are in place so implementation choices can be made to avoid problems (Albtoush et al., 2011). This also enhances the overall viability and feasibility of the system.

Emergency Systems are growing at fast pace (Alazawi et al., 2014). In contrast to (Benkhelifa et al., 2014), (Alazawi et al., 2014) focuses on Vehicular Ad hoc Networks

(VANETs), sensors, social networks and Car-to-X, where X can either be infrastructures or other cars. These technologies are shaping the future with the objective of giving a ubiquitous sensing of the surroundings. These systems produce large quantities of data, changing the context of looking at them from small, simple solving problems, to big data problems that require stronger and more capable algorithms (Alazawi et al., 2014).

2.5 Smart Emergency Systems

Smart emergency systems are an extremely important piece for the welfare and wellbeing of people. These systems provide computational ways of responding to dangers. When they are in place, the probability of anticipating man-made or natural disasters increases.

In (Radianti et al., 2014), the authors present emergency systems and then start to develop a platform that intends to mimic these systems in a smarter way. Figure 2 illustrates a smartphone based publish-subscribe system to accomplish this. The platform helps users by sensing their surroundings and assessing the current disaster scenario, providing them with a safer way to exit the building. It is interesting to analyse the communication that was developed as it takes the data from devices and delivers it, via a web-based broker, to managers and interested parties. The broker also forwards the data to a database where it is processed in order to retrieve sensor information in useful ways (e.g. charts, reports).

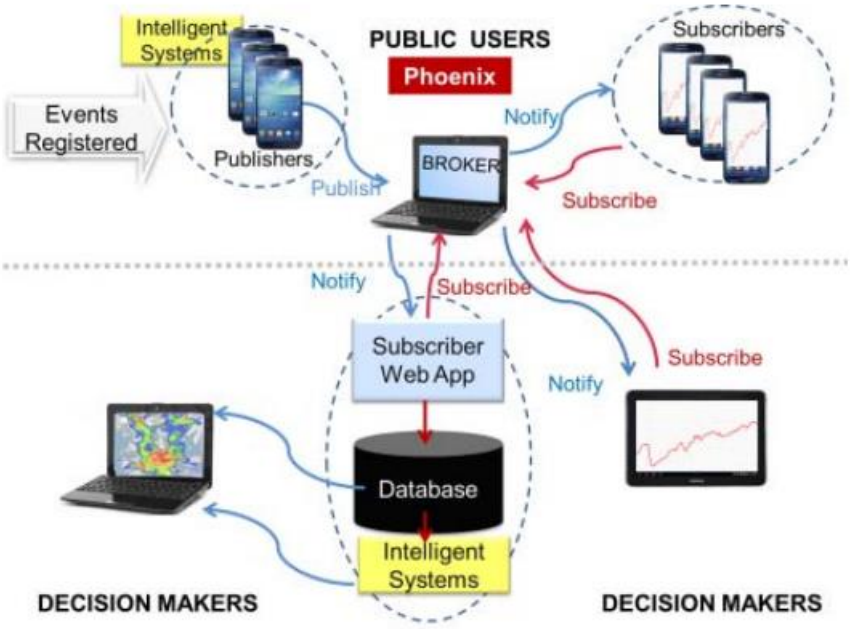

Figure 2 - Smart Rescue Platform (Radianti, Gonzalez and Granmo, 2014)

In our work, we intend to present an architecture for a generic smart system that collects, processes and delivers useful data to users. A smart emergency system will be developed

and will integrate information from many places, process it and then retrieve it to interested parties.

3. PROPOSED ARCHITECTURE

This chapter presents our proposal for an architecture for an IoT system. The background on the subject and the required technologies to implement a system using this architecture are also provided.

The systems already in place are decentralized, which means that they do not communicate between each other, making it almost impossible to prevent disasters. Most of the times these systems are designed to address a specific case or to work as an independent system that may receive information from many parts, although without the aim to deliver information to the necessary parties. With the intent to address these shortcomings, an architecture for a smart system in the context of smart cities will be provided. This architecture has been created with awareness of the system's possibility to scale and to adapt itself to different contexts. It will address the problem of receiving the data, process it and then retrieve useful outputs to any party that subscribes to a specific type of content. This architecture can then be tuned to fit different use cases and scenarios.

3.1 Citibrain

The architecture we are proposing will integrate Ubiwhere's Citibrain platform. This platform already provides solutions such as:

- Smart Waste Management – which intends to manage urban waste from a city with sensors placed in trash containers;
- Smart Water Management – intends to manage water leakages in a city, aiming to prevent, detect and correct these type of problems;
- Smart Traffic Management – intends to solve the traffic problem in urban environments.

Figure 3 illustrates the current offerings of the Citibrain platform. These offers are divided in five main areas, which are:

- Mobility;
- Environment;
- Monitoring;
- Payments;
- Connectivity and Interoperability.

The main purpose with this platform is to provide the necessary tools to empower a city and make it smart. Therefore, it can be divided in areas which intend to segment applications, regarding their characteristics and usefulness in specific use cases.

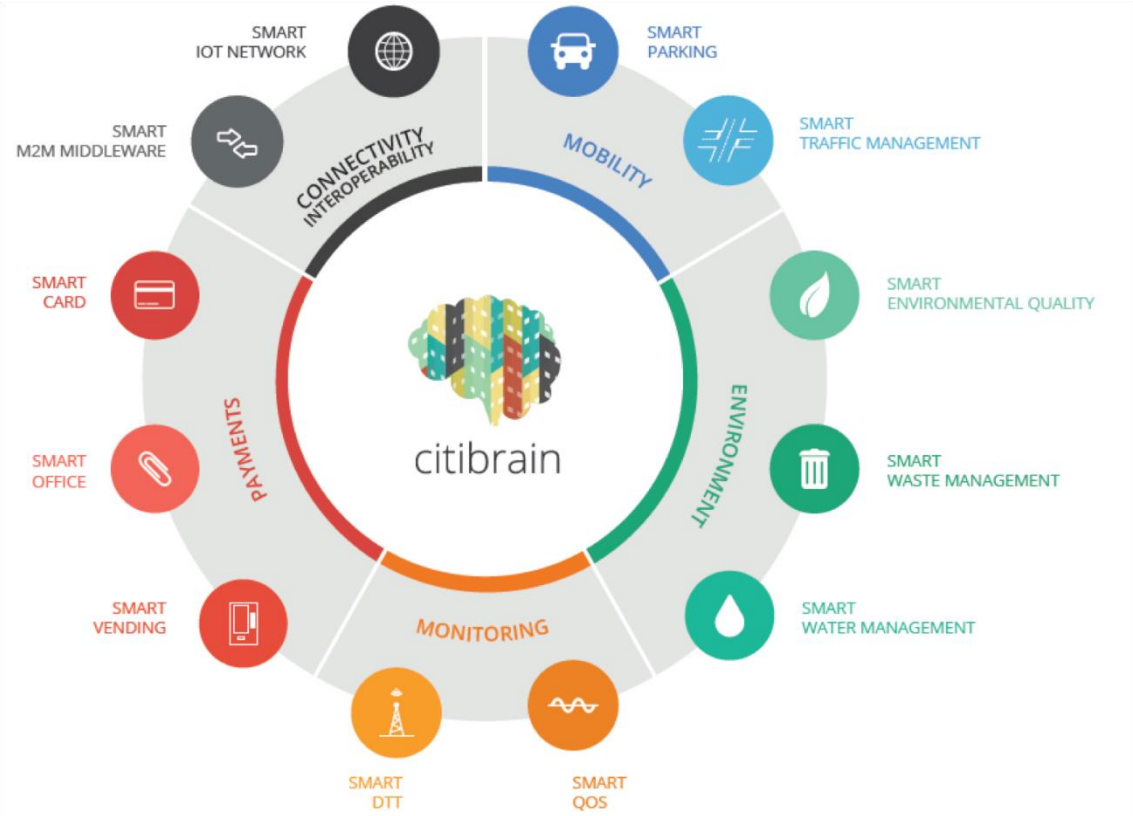

Figure 3 - Citibrain Platform (Citibrain, 2015)

The prototype we aim to develop will integrate the platform as a core component, providing unified access to alerts generated by other applications. For example, a user that subscribes the Smart Water Management Application and the Smart Waste Management Application will have access to a unified control centre. This control centre will monitor every application providing useful metrics from it.

3.2 Background

Systems in the IoT field require different technologies in order to be fully addressed, therefore this section aims to cover and introduce some of them which are, in some cases, the ones that have used.

Big Data is referred to as "(…) the processing and analysis of large data repositories, so disproportionately large that is impossible to treat them with the conventional tools of analytical databases" (Friess and Vermesan, 2013). The authors also explain that this data is produced by machines, which are much faster than human beings, and according to

Moore's Law this data will grow exponentially (e.g. web logs, RFID, sensor networks, social data, etc…) (Friess and Vermesan, 2013).

It is also referred that Big Data requires different technologies to process the massive amount of data within an acceptable amount of time, therefore some tools are presented in order to show the current standards in this field.

With the appearance of new technologies there is a new way of interaction between humans and the Internet via smart devices, which presents a challenge. This challenge exists because of the way that the Internet was created. Until now the Internet was based on a human to human kind of interaction, because it delivers content produced by humans for other humans. This kind of communication will not disappear, however new types of interactions will appear as smart objects integrate the nowadays world. These new types of interactions produce large amounts of data, which Big Data helps to store, with the objective of being analysed by intelligent algorithms and tools to extract information and provide knowledge. At this point it's possible to conclude that Big Data requires special treatment as it is larger and contains more information than typical data.

Regarding this topic, the authors in (Friess and Vermesan, 2013) explain that major companies in the big data topic have a tendency to use Hadoop (Gu and Li, 2013) due to its reliability, scalability and distributed computing. Hadoop is a framework that processes big data in a distributed environment (Apache Hadoop, 2014). The Hadoop framework (Gu and Li, 2013; Apache Hadoop, 2014) is planned to scale up from single to multiple machines, where each of them provides storage and computational power, therefore it is a good way to implement the system. However, in more recent works Spark (Gu and Li, 2013) started to be used instead of Hadoop. Spark (Spark, 2015) is a general purpose, in-memory, big data processing framework that provides APIs in Java, Python amongst others. It also provides other tools important for machine learning (e.g. MLib, SparkSQL).

It is important to understand that Hadoop is an implementation of the MapReduce framework developed by Google. Hadoop is not designed to support applications with iterative nature, as it cannot keep data during execution time (Gu and Li, 2013), because of this, at each iteration, it needs to access the disk. On the other hand, Spark, despite being a MapReduce-like framework, is designed to address its current shortcomings regarding iterative applications. Also it is an in-memory technology, which allows for faster performance.

Finally the authors concluded that both frameworks are good, but their application depends on the situation. If there is a lot of memory to run the application, Spark is definitely faster than Hadoop, on the other hand Hadoop uses less memory but much more space in disk.

Other types of data processing are also interesting in the Internet of Things (IoT) context, due to their ability of processing data streams. For instance we can point out Complex Event Processing (CEP) (Chen et al., 2014) and Storm (Toshniwalet al., 2014). Notice that CEP is only a method of analysing and processing streams of data, while on the other hand Storm is a distributed computation framework that helps with the processing of large streams of data.

CEP is defined as an effective mechanism that analyses data and its context to trigger events (Chen et al., 2014). CEP can, for example, analyse streams of temperature and determine if changes in that temperature are normal or abnormal and can also relate different types of events that lead to a single complex event, such as: (1) flames; (2) temperature spike; (3) sudden humidity decrease. From these three events the system could infer that a fire was happening. Additionally (Chen et al., 2014) aims to develop an architecture for the IoT based on distributed complex event processing. The intent behind distributed CEP is to shorten the bandwidth and the necessary computation for event detection. The leading tool for CEP is Esper (Esper, 2015) which is an open source Java implementation of a CEP engine, which allows for real time stream processing.

Storm (Toshniwalet al., 2014) is a real-time distributed data processing and stream data processing engine that manages data streams. It was designed to be scalable, resilient, extensible, efficient and easy to administer which makes it a very robust and usable structure. Figure 4 presents a storm topology, which is the real time component that runs all the logic. Topologies are then divided in spouts and bolts. Spouts, represented by the water tap in Figure 4, and are the source of the streams of data. Bolts, represented by bolts in the topology, intend to consume the data sent by spouts, process it and then produce processed outputs.

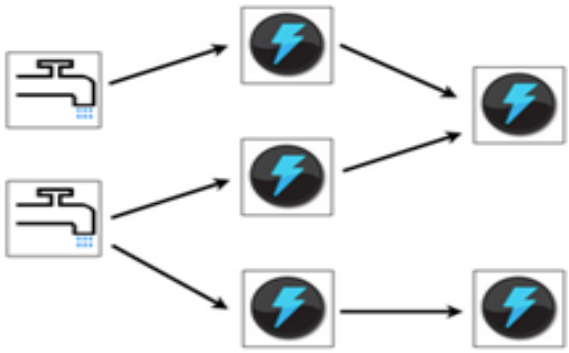

Figure 4 - Storm Topology (Apache Storm, 2014)

Furthermore, Figure 4 illustrates a fault tolerant and scalable architecture for handling data (Apache Storm, 2014). Additionally, this architecture provides the concept of worker that can be interpreted as a node which is programmed to execute a specific task. These tasks may vary, although a good example can be the use of a worker to process the stream with the Esper queries, which are statements similar to SQL which allow the processing of events in real time.

Additionally, Esper and Storm can help one another: Esper needs something to organize and provide data, which means that some system needs to be implemented to provide data to Esper. This is where Storm can be useful as it can handle the data management while Esper handles the queries. This approach will join both systems to enhance both of their main capabilities when dealing with these types of data.

To access the data from cities, sensors and other devices are required. These devices and the communication protocols are comprised in the concept of Machine-to-Machine (M2M) (Wan et al., 2012). M2M refers to the automatic communication between computers, sensors and other devices in the surroundings (Wan et al., 2012). This topic is relevant because it makes sensor-to-server communication and sensor-to-sensor possible. This allows the system to constantly check for new data and vice-versa. This concept leads us to publish-subscribe services. According to (Ordille et al., 2009) these services broadcast information to the subscribed parties and in these types of systems, a subscriber is a device that will receive information from the publisher. This translates into a much more transparent system, because the publisher can send information to the subscribers and vice versa. In (Radianti et al., 2014) the publishers are treated as the ones that generate information in the form of events, subscribers are treated as the ones that

subscribe to arbitrary flows of information and brokers are a middle layer between the two participants to pass along the information.

3.3 Architecture Example

In this subsection, current use cases of similar systems will be addressed. This will result in a better knowledge base for the current standards in the area. For this, not only examples of smart cities will be presented but also examples of emergency systems that became smarter with the inclusion of these new concepts.

Beyond the systems studied in chapter 2, one more was analysed. The system which was analysed was the SMARTCAMPUS that aims to equip the SophiaTech campus, in France, with sensors to inspire the creation of new applications (Cecchinel et al., 2014). Once more the system was chosen due to its usefulness and value in terms of possible inputs for our system.

SMARTCAMPUS deals with many types of sensors to collect the data. To tackle this challenge the authors propose the architecture illustrated on Figure 5. This architecture is divided in two main focal points: the message collector which intends to collect all data from the Internet or sensor networks, to further store in a database that acts as a message queue; and the message processing that aims to process the messages stored in the queue. These components then store the processed information in a database. Furthermore the architecture contains a configurator, which acts as a routine that can be called periodically to propagate a specific sensor configuration through the network. It also contains a database that contains the current sensor parameters, an API to provide an administrator interface to connect with sensors and a data API that directly accesses data to provide statistics or other types of knowledge.

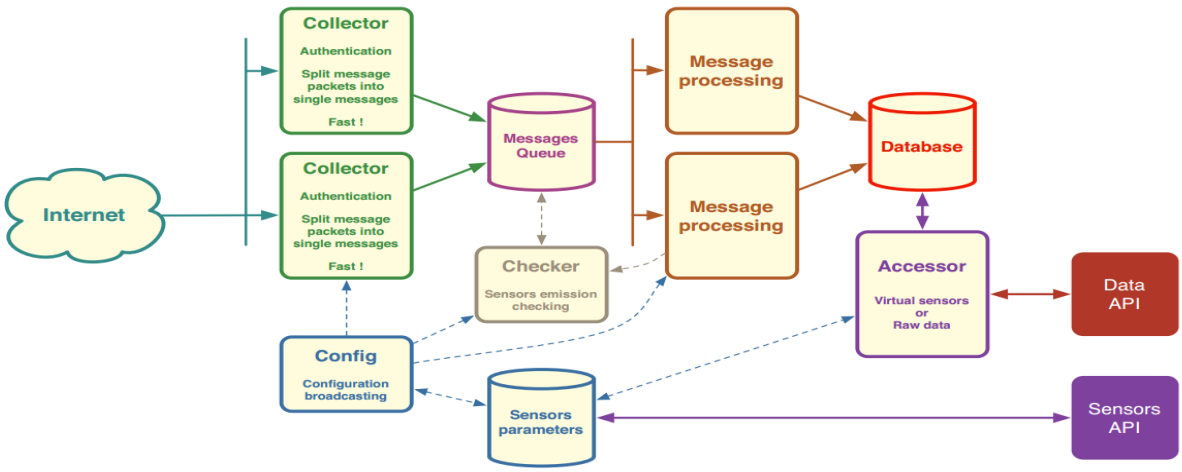

Figure 5 - Middleware Architecture (Cecchinel et al., 2014)

3.4 System Architecture

After gathering all this information it is important to clarify the requirements for the system we aim to develop. These requirements were assembled by reviewing the state of the art systems. Below a list of requirements will be presented:

- Handle, process and store streams of data from sensors;
- Generate alerts from the incoming stream of data;
- Generate predictions from history data;
- Provide KPIs (Key Performance Indicators) and historical data;
- An API for users and developers to connect.

Thus, we have come up with a proposal for an architecture. This architecture will provide a way to gather information from many sources, process it and provide useful information to the interested parties.

One of the most important things to understand is that nowadays data comes mostly in streams, which presents an issue due to the tools needed to process it. The tool that is projected to be used is Storm, which has already been described. Even though Storm, by itself, cannot retrieve the most accurate results in real time, due to the processing time needed, it is planned to overcome this problem by implementing a parallel processing block with Hadoop. This will not only provide exact results when the large amount of data is processed, but also provide a better knowledge of the data.

The approach was inspired by the lambda architecture (Lambda Architecture, 2014) with the publish/subscribe pattern. The background from other related projects allowed us to

perceive that some technologies may not suit very well the collection and direct processing of data. Thus, we opted for a more complex approach that allows a more scalable and reliable system.

This type of approach also led us to extend the capability of receiving data from multiple sources, which is extremely important in the context of IoT.

In Figure 6 our proposal for the architecture is presented.

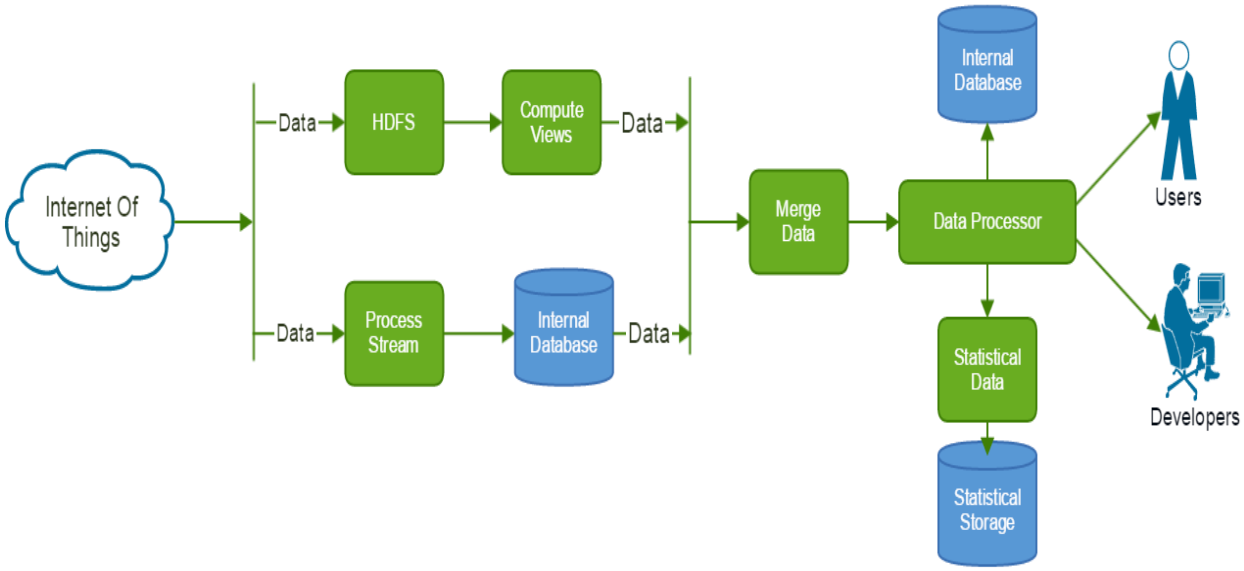

Figure 6 - Proposed Architecture

Our architecture is projected to act as an API to provide a connection between data in the IoT and the end user, with the intent of providing relevant information regarding emergency situations.

The system will receive a data stream from IoT nodes, which is then split in two parts to be processed by the batch layer, responsible for demanding calculations and the speed layer which delivers results in real time. After that, the data is merged with the intent of providing the result with the biggest confidence level associated. When the data is merged a bottleneck can happen, although this situation will be prevented by accepting the first result to appear with the highest confidence level. This can happen in two ways: (1) the stream layer finishes and the batch layer continues to process; (2) the stream and batch layer finish almost at the same time. In the first scenario the stream layer result will be

returned with a confidence level attached to it. In this second case the data will be merged to provide the most accurate output.

After the data is merged, it reaches another processing block, which intends to filter and redirect the acquired knowledge to the subscribed parties. Additionally this block sends the processed data to the statistical data block. The latter block not only keeps track of statistical data to help understand patterns along the year but also provides data to construct KPIs, charts and reports.

After all the processing is done, users can access the data in two ways: (1) via the API, which is projected for developers who want to build applications around this context; (2) via the data output, which will serve to return the data to the subscribed parties. Additionally, the API will provide a way of notifying other sensors, which means that if a sensor sends an alert, other sensors around it will be asked for their current situation to locate the hazard with maximum precision. This type of communication is also important if the alert is located, for example, near a road, since the system can be prepared to notify street lights to prevent drivers from entering the affected road and in highways a lane can also be closed being the traffic redirected to other lanes or roads.

3.5 Use Case

Our architecture can be applied in many different scenarios; one of them will be addressed so that we can establish an example to explain some of its functions.

Figure 7 illustrates a simple example of a possible use case. Let’s assume we have three types of sensors: smoke, flames and temperature. These sensors are constantly sending streams of data to the system and the idea is to process this data in order to figure out whether we are in the presence of a fire or not. The system has a threshold that serves as a maximum possible value for a normal event, when crossed they trigger events that can lead to an alert.

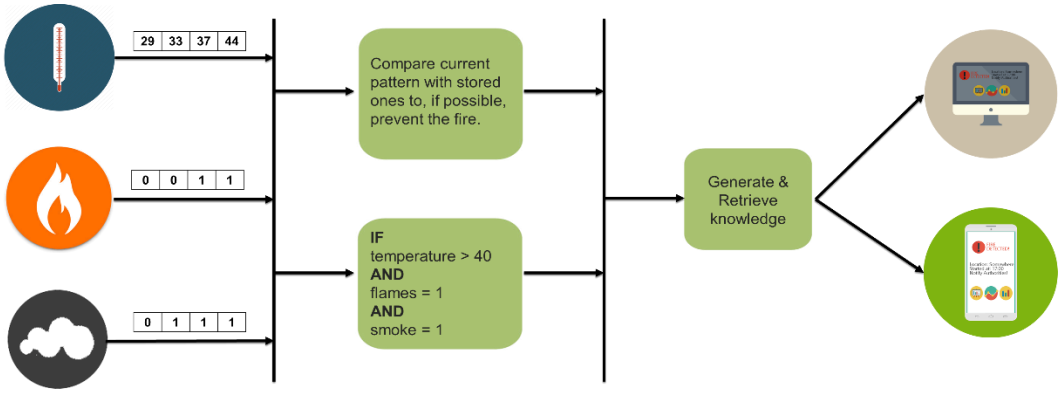

Figure 7 - Example application

Having different types of sensors allows us to better understand whether the fire is happening and different combinations of events can occur, thus the system must have something to divide the ones that are indeed problematic. Furthermore an example shall be presented:

- If there is smoke, flames and the temperature passes the threshold, then we have a fire;
- If there is smoke, no flames and the temperature is rising, it is possible that we will have a fire.

Many more combinations can be presented, although these explain the concept that we are trying to achieve.

On the IoT data comes mostly in streams, hence we need to account for the data stream that is arriving. For instance, the architecture should use a publish-subscribe messaging system, which handles the stream and splits it into events that can be processed by the rest of the modules. The events that have been split will be processed by both layers. At this point, in the speed layer, there are two important things to acknowledge: (1) it is advised to use a complex event processing tool due to the event driven nature of the system; (2) a database with high write speed for storing alarming events is also useful, because of the high demand from the system. This will provide an event based approach which will detect event correlations and deal with the data stream that is constantly changing. This approach will also provide the ability of integrating many types of events at once, this will expand system acceptance in terms of receiving events and inevitably prepare it to explore further sensor integrations.

In the batch layer, algorithms with predictive capabilities should be added to enhance the system overall quality and usefulness. This will provide ways to calculate KPIs, draw

charts and predict whether it is important or not for emergency response teams to be prepared. From a high level perspective this type of inputs seem to have a great importance, such as divide a specific fire protection team to a zone which is prone to peaks of fire during the summer or redirect traffic because a particular road is more likely to be affected by the floods in the winter.

The rest of the components do not need to be a specific technology, although we point out some advices for when choosing the technologies to work with.

The processing components in the system can be executed with any programming language and should withstand the volume and velocity of data, also the code should be optimized to minimize overheads and bottlenecks. The database should be chosen according to the needs of each specific scenario. It is important to understand that many database systems can be chosen to incorporate the solution, although for each specific situation a brief analysis of the problem should be made in order to perceive the best possible choice. As a practical example we can point out that the database in the speed layer should be in-memory due to its velocity, while on the other hand the statistical storage could be a NoSQL database that supports large quantities of data to enhance overall system scalability.

Moreover other important aspect to discuss is the communication. The way the system is designed, and from the lessons learned from the use cases (e.g. Santander city systems), the best technologies should be REST, WebSockets and AMQP. REST will provide an easy and consistent way to access the API, providing endpoints for events and the ability to execute filters in the queries; WebSockets are useful due to their ability in terms of real-time communication and the AMQP protocol is important to establish connection between the system, sensors and actuators scattered in the city in order to extract information.

Additionally, another important aspect is the inclusion of a message broker, which will accept raw data from the source and divide the stream in messages that are easier to process and correlate for a more useful and more accurate output, which is delivered to a consumer.

3.6 Technologies Used

In this section we intend to list and introduce the technologies that are needed to develop this architecture. Thus, the list bellow contains all of the technologies used to develop the system:

- RabbitMQ (RabbitMQ, 2015) – This is an open source messaging broker, which implements AMQP (Advanced Message Queueing Protocol). It was written in Erlang language which allows it to guarantee messaging failover. RabbitMQ is currently owned and maintained by Pivotal. RabbitMQ provides two different ways of communication, which are queues and exchanges:
    - Queue – A queue is a mechanism that provides asynchronous communication between a sender and a consumer. It can be seen as an infinite buffer of messages that await processing;
    - Exchange – An exchange is a mechanism that allows queues to be connected and to receive the events that are sent to it. This is an important concept because it allows the consumer to connect multiple queues.

  In our prototype RabbitMQ is the messaging system used for all communications.

- Meshblu (Meshblu, 2015) – This is an open source M2M messaging tool that allows data from the sensors and machines to be sent in an understandable way. It is used in our prototype to send the data from sensors to RabbitMQ;
- REST (REST, 2015) – Although not considered a technology, REST is very important so it was decided to include here. REST is an architectural style that allows web services to scale and communicate. It uses the HTTP verbs such as GET, POST, and DELETE. REST is used to expose an API, which handles the system's overall functions;
- Java (Oracle, 2015) – It is an object oriented programming language that allows developers to create robust and secure enterprise applications. It is known for allowing to "write once, run anywhere", as it is present in desktop, mobile and even web applications. This programming language is used to build almost every component of the prototype;
- Spring (Spring, 2015) – It is a framework that delivers a set of programming and configuration modules that allow developers to abstract themselves from the programming language and focus on the problem. In our work this framework

facilitated web development with Java. It is used to manage REST calls, dependency injection and HTTP responses;

- Socket.io (SocketIO, 2015) – It is a JavaScript library to implement WebSockets on real time web applications. Therefore, it enables real time, bidirectional communication. It is very popular because it has multiple fall-backs which guarantee the delivery of the message. In our prototype Socket.io is used to guarantee WebSocket connections to the generated events;
- Esper (Esper, 2015) – It is an open source Java implementation of a Complex Event Processing (CEP) engine, which allows for real time stream processing. For the stream processing it uses Event Processing Language (EPL) which is very similar to SQL. Esper is, in our prototype, the chosen CEP engine to generate events from the incoming data;
- Node.js (NodeJS, 2015) – It is an open source runtime for server-side JavaScript code. It is widely used in real time web applications due to its event driven architecture. Furthermore it uses the Google V8 engine (Google Developers, 2015), which compiles JavaScript to machine code. Node.js is the second programming language used by our prototype. It was used to create the WebSocket server and can be used to create the clients as well.

In the next chapter we shall discuss the database that was used in the system.

4. SYSTEM DATABASE EVALUATION

In this section we describe the NoSQL database Cassandra, chosen to integrate our system. To the best of our knowledge, a perfect solution for the Internet of Things data layer is yet to be found. With this in mind we aim to find the best possible solution for these type of systems. Thus, we started exploring which database would be the most suitable to provide a production ready environment. The database will be used in an Internet of Things system which needs to be production ready and receive enormous amounts of events in real time. This system intends to gather data from a city and process it in order to find events that are considered dangerous.

4.1 Overview

Systems on the IoT scope that deal with sensors is becoming gradually difficult to scale due to the amount of sensors and clients that extract data from them (van der Veen et al., 2012). Therefore it is important not only to pay attention to the velocity of the data, but also to the probable volume that it will gain during the time the system is deployed. With the inclusion of social mining it can even reach new dimensions in terms of volume and variety.

According to (Abramova et al., 2014a, 2014b, 2014c) Cassandra seems to have a clear advantage in terms of the needed characteristics to be implemented in our system. "Cassandra system was designed to run on cheap commodity hardware and handle high write throughput while not sacrificing read efficiency" (Lakshman and Malik, 2010), also the decision of choosing Cassandra is related to the market share and popularity (DBEngines Ranking, 2015).

In addition to storing data, every system needs to provide it in order to query and filter later. It is important to understand that we will be receiving events from external applications which are registered to our system. Although the usage of Cassandra with these reading characteristics seems sub optimal, the main focus is the insert rate and on that Cassandra does a great job (DBEngines Ranking, 2015). It is important to keep in mind that systems included in the IoT context tend to be stream oriented, rather than batch. For this reason, the database to be chosen needs to accept data in streams, or at least support a high rate of data insertions, and have the necessary mechanisms to withstand this.

The main reason for this analysis is to understand which architecture for the data layer would best suit the needs of an IoT platform in terms of querying performance, without sacrificing the write speed. There were two ways which we have thought would be relevant in terms of implementation. First, a single table with all the data, which would then be filtered and dealt with when needed. A second approach is multiple tables for each specific application that sends events. From a theoretical standpoint it seems that the best way of organizing our data is through the creation of a table per application. This will result in smaller tables which, in comparison to a centralized table that stores everything is a lot faster because they have significantly less records. In the coming sections some experiments were documented for better understanding. Figure 8 illustrates the two cases, where on the left we can find the single table and on the right the multiple table configuration.

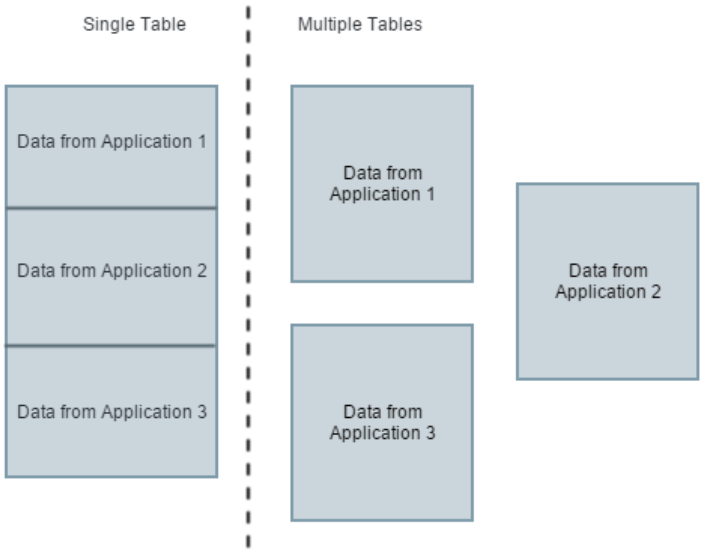

Figure 8 - Data layer possible architectures

4.2 Cassandra

Cassandra is a distributed storage system that manages large amounts of data across servers (Lakshman and Malik, 2010). Still according to the authors Cassandra uses a combination of well-known procedures that grant scalability and availability.

4.2.1 Data Model

Cassandra's data model provides a high processing speed when writing the data, this is due to the indexing.

Cassandra indexes data by key, this key is a unique representation of the row which contains the data. Each row contains columns, which are attributes and finally these columns make up a column family.

Figure 9 illustrates the data model, which is composed by rows, column families and keyspaces.

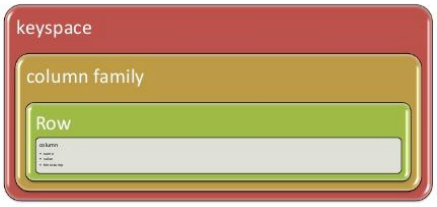

Figure 9 - Cassandra's Data Model (Charsyam, 2011)

Furthermore we shall address the two important concepts which make up the data representation in Cassandra, which are the column families and the keyspaces.

- Colum Family – A column family is a container for a group of rows (Hewitt, 2011). Column families are not defined, which means that the structure can be changed at any desired time, this improves the system's readiness to change and adapt during time;
- Keyspace – In Cassandra the keyspace is the equivalent to a database in the relational paradigm. The keyspace contains the column families which make up the full database. The keypaces contain attributes that can be tuned to enhance the overall performance of the database, these attributes are:
    - Replication factor – which refers to the number of physical copies of the data. For example if the replication factor is set to two data will be replicated twice;
    - Replica placement strategy – this attribute is used for defining the strategy of how data is placed in the cluster. There are some possibilities to define the replicas. As examples we can point out the SimpleStrategy which is most used when we have a single group of nodes in the cluster and NetworkTopologyStrategy which is more used when the cluster is working across multiple machines providing a way of managing the replicas in all the machines.

Lastly, Cassandra provides the notion of Super Column Families which are useful to define new types of data or more complex data structures which are not yet defined by the default types. The Super Column Families are organised in Super Columns which contain a name and the new columns that are needed. For example, if a new type "Address" is to be defined the Super Column should contain the new type name, in this case "Address" and then a key-value map which contains the attributes of the type (e.g. "Street", "Street Name").

4.2.2 Architecture

In this section is given an overview of the Cassandra architecture. Cassandra uses a peer-to-peer architecture, which means that all nodes within a cluster can receive a request and respond to it (Strickland, 2014). This provides better availability when the database is online. Also, this provides redundancies which help to keep the data safe and horizontal scalability. In Figure 10 we can observe the Cassandra peer-to-peer architecture.

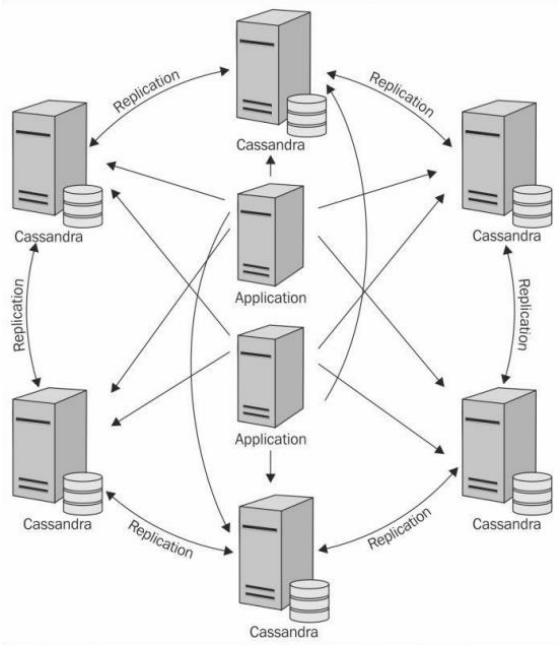

Figure 10 - Cassandra Architecture (Strickland, 2014)

Furthermore, this architecture provides high availability to the database, this means that the system does not have a big downtime period, providing constant access to the data.

4.2.3 Replication

Replication is very important in Cassandra because it provides ways of copying the data within or across nodes. This is done by storing the replicas on the keyspace they belong to.

Cassandra provides two different replication strategies:

- Simple Strategy – This strategy is normally used for single data centre deployments (Datastax, 2015). When this strategy is done, Cassandra will find the first replica and then will perform a clockwise movement to store the next replica. When creating this strategy the number of replicas must be defined. Figure 11 illustrates this strategy. The first replica is the original inserted value, the rest are copies placed in a clockwise fashion to replicate the data. The replication factor used was 3.

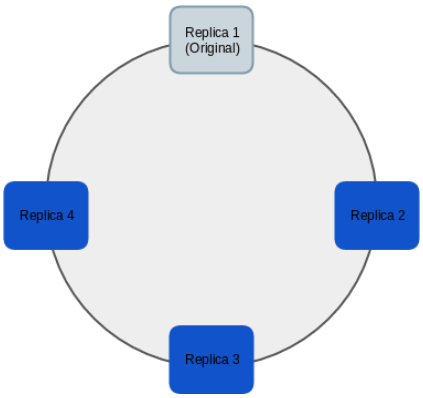

Figure 11 - Simple Strategy

- Network Topology Strategy – This strategy is used when the cluster spans across multiple data centres (Datastax, 2015). It places the replicas the same way as the Simple Strategy, although it places them in different physic groups (racks) to enhance the safety of the data in case of sudden crashes. When creating this strategy the number of replicas and the number of data centres to keeps those replicas must be defined. Figure 12 explains this strategy by providing an example. This example creates the copies in two different Data Centres with a replication factor of 3.

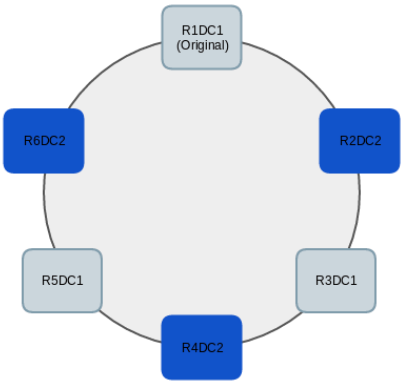

Figure 12 - Network Topology Strategy

4.2.4 Writing and Reading

Cassandra is a Column Family NoSQL database, which translates into a data format storage which is vertical oriented. The appropriateness of this database for logging systems (Abramova et al., 2014a), led us to acknowledge that it could be used in IoT.

Additionally (Abramova et al., 2014a) provides an architectural overview stating that Cassandra divides each received request into stages to enhance the capabilities while handling and serving a high number of simultaneous requests. This allows Cassandra to improve its performance, however it is limited by the host machine characteristics, mainly by the memory available. Finally, and because RAM memory is a lot faster than the standard HDDs and SSDs Cassandra needs to have a mechanism that will handle writing all this data to disk, in background. According to (Abramova et al., 2014a) this mechanism is called memory mapping and consists in two similar mechanisms: the Key Cache and Row Cache. Key Cache handles in memory mapping of the stored keys and it's solely responsible for storing in RAM these keys, providing fast read/write speeds on them. On the other hand, Row Cache is the memory mapping for each row.

To better demonstrate the life cycle of a record being written in Cassandra we will provide an overview of the writing architecture. Figure 13 explains how Cassandra writes a record. First it writes every arriving row in the commit log, then it replicates this data on the Memtable. The data is replicated in the commit log to ensure that there are no records lost. The data which is now on the Memtable will only be written to disk when a flush happens. A flush can happen when: (1) it reaches the maximum allocated memory; (2) after a specific time in memory; (3) manually by the user. When flushed, the Memtable becomes an immutable Sorted String Table (SSTable) which stores all the data (Ordille, Tendick and Yang, 2009).

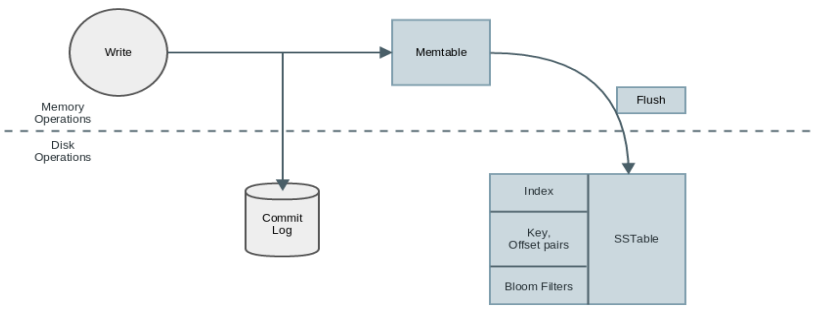

Figure 13 - Cassandra Writing

Figure 14 explains how Cassandra reads the data within one cluster. A request is made to any node in the cluster, the chosen node will become responsible for handling the requested data. The request is then processed and all the SSTables for a specific column family will be searched and the data will be gathered to merge data. Merge data is useful because of the replication factor of the tables, for instance nothing guarantees that the data is all stored in the same table. When a read request is made it might need to gather data from multiple tables, merge data allows this data to be combined.

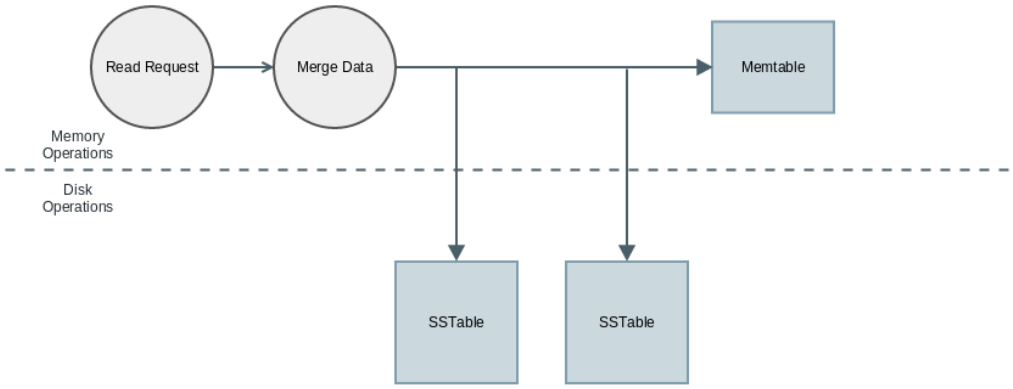

Figure 14 - Cassandra Reading

4.3 Experimental Setup

The experiments that will be made will allow to learn which approach is better when storing data in the IoT. As mentioned before we have decided that there were two ways to organize the database which would be relevant in terms of implementation. A single table with all the data, which would then be filtered and dealt with when needed, or multiple tables for each specific application that sends events.

The experimental setup was created with the following characteristics: (1) The operating system was Ubuntu 14.04 LTS 64bit; (2) The machine had a dual core, Core i5 480m with 6GB of RAM and an HDD; (3) Cassandra ran in a single node to understand the minimum possible requirements when running the system.

We have decided not to use a benchmark tool because we have concluded that most tools available nowadays do not provide the necessary requirements to test the database system with the necessary characteristics. Also with this approach we guarantee that the performances we see are more accurate and can be replicated in a production environment.

The chosen queries intend to illustrate regular situations during the usage of the system, which reflect the better approaches to the problem, keeping in mind that attention to the

write speed is also needed. To analyse them, different queries will be created, matching the needs while the system is in place. These queries may vary from time to time, although some of them will be a recurrent task that needs to be performed. Additionally, it is important to keep in mind that these queries are to be performed in an IoT system, which generates alerts with the data that comes from the sensors scattered around a city. The idea is that these alerts are filterable and searchable throughout the lifecycle of the system.

In the experiments we have the following queries:

Q1: Alert selection from a specific type – This query is performed to provide the number of alerts of each type (e.g. Number of 'warning' alerts);

Q2: Alert selection for a submitted rule – This query will be used to see how many alerts were raised by a submitted rule (e.g. how many alerts were generated by rule X);

Q3: Alert selection in a range of time – This query serves to select a type of alerts (e.g. 'warning', 'critical') in a period of time.

These queries give a broad perspective of the system in terms of querying performance.

To query the database we use the Cassandra CQL shell, to record the times we have enabled tracing which allow us to have a detailed view of the query and created indexes to allow filtering to happen. Figure 15 shows the row prototype.

Figure 15 - Row prototype

The row is composed by the following columns:

- alert_uuid – This field is of the type UUID, it represents the universal id of the alert to keep each alert unique;
- config_id – This field is of the type UUID, it represents the application id which created this alert;
- event_query – This field is of type TEXT and it represents the rule needed to fire the alert;
- alert_type – This field is of the type VARCHAR and represents the type of alert which was generated (e.g. Critical, Warning);

- event_type – This field is of the type VARCHAR and represents the type of event to be processed (e.g. Environment, Traffic);
- event_window – This field is of the type TEXT and represents the event window which triggered the alert;
- event_body – This field is of the type TEXT and represents the full event which triggered the alert;
- created_on – This field is of the type TIMESTAMP and it represents the timestamp on which the alert was triggered.

On the next section we will present the results of the experiments.

4.4 Query execution evaluation

In this section we evaluate the query processing time. Each chart contains, in the Y axis, the "Query Time (ms)" which represents the time the queries took to be processed. In the X axis, we have "Table Name" which represents the table where the query was made. The tables are divided by configuration and each represents an application. The "Table Name" axis uses the following notation:

- **App1-App5**: correspond to applications with data that comes from environmental sensors. Each of these applications have 100.000 records;
- **All**: corresponds to the single table containing all the information. This table will have 500.000 records.

The values presented in the experiments were obtained by executing the same query five times and then calculating the average value. Also, the first three queries of each run were discarded due to the possibility of cold boots. In the figures the dots represent the average value of the query speed and the error bars represent the standard deviation to that value.

For a better approximation of a real system, the queries were made in no specific order. This has to do with the Cassandra reading architecture which is faster if the table is in memory.

In the next sections we will show the values obtained during the experiments and present a summary of the values obtained.

4.4.1 Querying an alert of a specific type (Q1)

In the experiment we use this query to select all the alerts of type 'warning' from the applications. Using the CQL language the query looks like this:

```
SELECT * FROM query_performance.alerts_<config_id> WHERE alert_type =
<alert_type>;
```

For the table with all of the data the query used was:

```
SELECT * FROM query_performance.alerts_full WHERE config_id =
<config_id> AND alert_type = <alert_type> ALLOW FILTERING;
```

This a very simple query, since it only lists the alerts of type 'warning' that were generated by the application. However it is expected to see an enormous change in terms of performance, due to the amount of data in the "All" table.

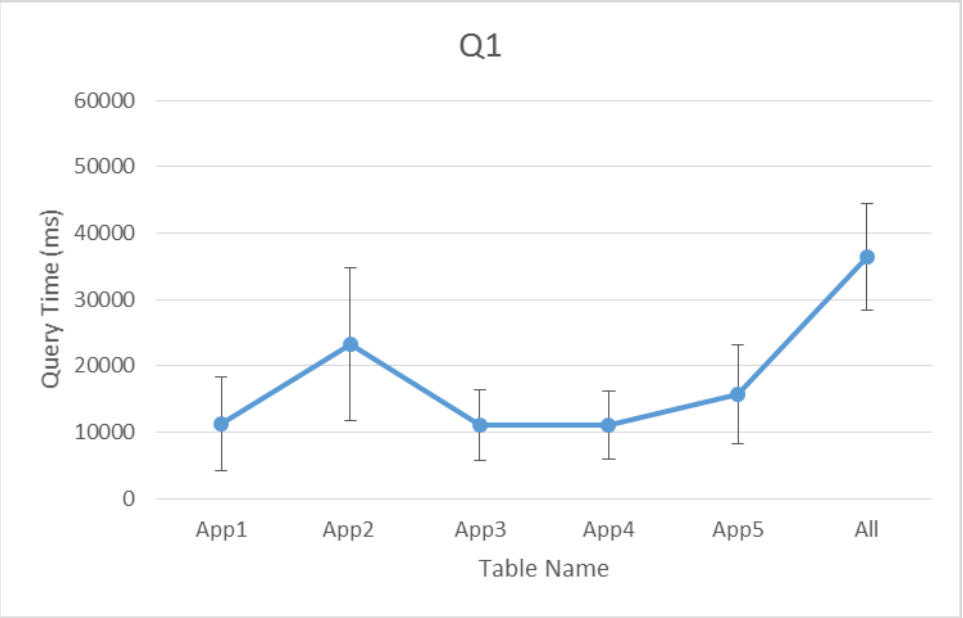

Figure 16 – Query execution time of Q1

When analysing the results of Q1, present on Figure 16, we can conclude that the separate tables were, in general, the best choice. Although in the second application we saw a little deviation from the average value, this is related with the reading architecture of Cassandra which is faster if the table is in memory. As explained before, we have tried to make queries to different tables in order to provide results which are useful for people who want to know if this database is a liable option for a production system.

4.4.2 Querying an alert for a rule (Q2)

This query intends to list every alert for a specific rule created by the user. The query, using the CQL language, will look like this:

```
SELECT * FROM query_performance.alerts_<config_id> WHERE event_query =
<rule>;
```

For the full table the query looks like this:

```
SELECT * FROM query_performance.alerts_full WHERE config_id =
<config_id> AND event_query = <rule> ALLOW FILTERING;
```

The query on the full table could not be completed because the operation timed out. The operation quitted when filtering the data with the where clause, this is due to the amount of data it needed to filter. We have tried to change the environment settings for Cassandra to try to overcome this situation, but the error persisted. This led to the removal of this query from the charts. Due to this problem, the comparison was made only between the applications. Furthermore, we can conclude that this query cannot be made in a production environment because the system cannot be stuck waiting for the query to end. On a real world system, and because IoT systems require near real time responses, it is impossible to implement this query because of the error it kept raising.

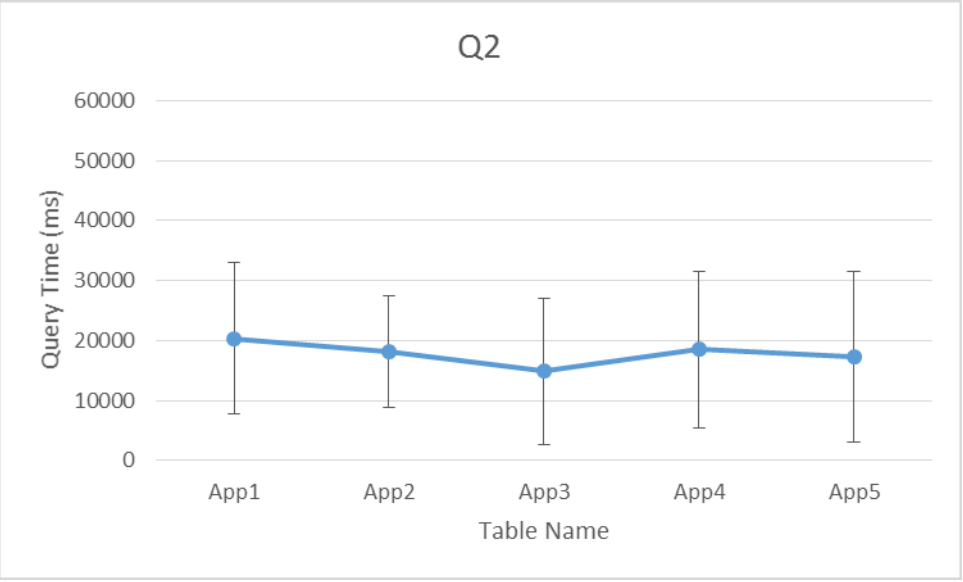

Figure 17 - Query execution time of Q2

With the results of the execution of Q2, seen on Figure 17, we conclude that every application has similar performances when dealing with this query. The main conclusion

to draw from this experiment is that the table with all the data could not be queried because it kept raising an out of time error. This is due to the amount of data which is stored in that table which Cassandra cannot filter.

4.4.3 Querying an alert on a time range (Q3)

This query selects all the alerts of each application in a time range. In the real system this query is important because it delivers a query that provides a time based approach to the data. Using the CQL language the query looks like this:

```
SELECT * FROM query_performance.alerts_<config_id> WHERE created_on <=
<timestamp> AND config_id = <config_id> ALLOW FILTERING;
```

The query made on the table with all of the information will look like this:

```
SELECT * FROM query_performance.alerts_full WHERE created_on <=
<timestamp> AND config_id = <config_id> ALLOW FILTERING;
```

This is a simple date query, it only filters data by timestamp. However, it is expected to see an enormous change between the applications and the "All" table.

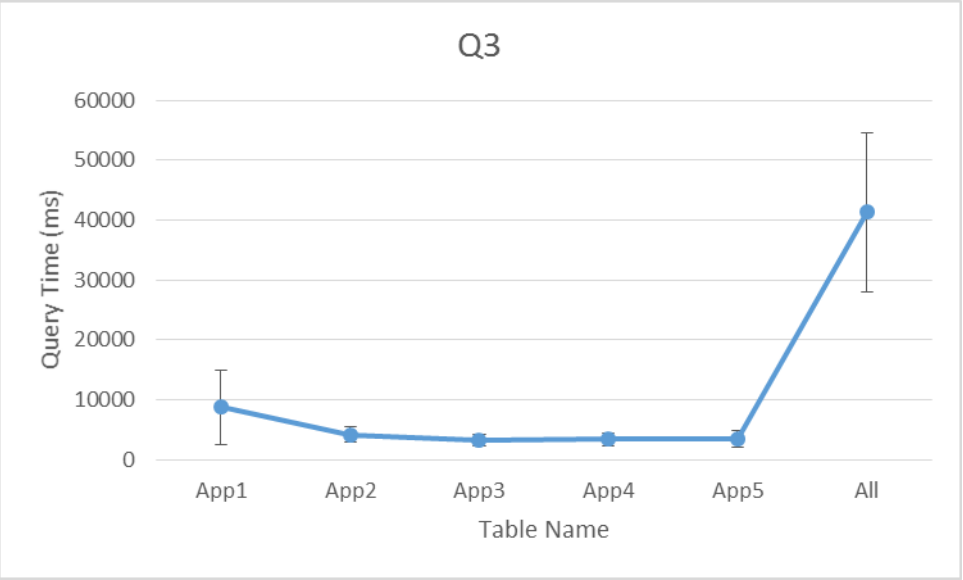

Figure 18 - Query execution time of Q3

The query Q3, in Figure 18, had comparable performance across all of the separate tables, the standard deviation on the first application is more due to discrepancy between the performances of when the table is in memory and needs to be loaded to memory. We can also see that the average time for the table with all the data is much higher than the others, once again proving that an architecture where the data is separated is better.

4.5 Summary

The results show that, as expected, the single table had the worst performance. This is due to the amount of data that Cassandra has to filter, which cannot be placed in memory all at once. Although the results of the "All" table were not five times worse we conclude that the best implementation is with separate tables which not only give a better performance, but also provide a better overall data separation.

The performance changes between the first two applications are a little bit different, this might be due to the size of the string that is being searched. The main differences are between the "All" table, which was finished on Q1 but not on Q2. This is due to the fact that, on these tables, data is sequentially organized which means that if the query results are not on the first records, Cassandra cannot load all the data to memory and initiate the filtering process.

The average query processing time in Q3 is a lot less than on the others, this is related to the fact that the dataset is not heterogeneous enough in terms of dates because the values of the applications were recorded on a single day. Also, filtering is made by primary key because in Cassandra to make a time range query the column with the date needs to be on the primary key of the table.

In short, we think that these queries, although very straightforward, give a quick and simple performance overview to a data layer architecture in the IoT. The results show that the single table had the worst performance. This is due to the amount of data that Cassandra has to filter, which cannot be placed in memory all at once. From this, we conclude that the best implementation is with separate tables, which not only give a better performance, but also provide a better overall data separation.

5. ALERTS MODULE IMPLEMENTATION

In this chapter we provide an overview of the alerts module implementation. This module is responsible for processing the incoming data from the sensors, process it in real time and raise alerts for each user. It was decided that the most suitable technology for our needs was Complex Event Processing (CEP) (Chen et al., 2014; Itria et al., 2014) and for the implementation of the CEP engine the chosen project was the Java based Esper (Esper, 2015).

5.1 Alerts Module Overview

According to (Itria et al., 2014) CEP "consists of the processing of events generated by the combination of data from multiple sources and aggregated in complex-events representing situations or part of them". CEP is a very interesting system in the IoT world because it can process streams of incoming data and be aware of events that can be specified with the usage of rules.

We have used Esper for the implementation because it is the leading CEP software in the open source market. Also, Ubiwhere already had projects which used the Esper engine, therefore we concluded it would be a good idea to follow up on a tool which has been used and is well documented. Esper is an open source software written in Java used for CEP. It analyses series of events and provides meaningful conclusions from them. It uses a standard language for building rules. These rules are called Esper queries and can be made with Event Processing Language (EPL) (Esper EPL, 2015), which is very similar to SQL but contains some additions that allow it to create time windows (e.g. an amount of data given in a confined range of time) in the data stream. Also, it is possible to specify a pattern that will return an event when the query is activated.

Our approach to the Esper system was made by recurring to a listener, which is instantiated for each application registered. This listener is then able to raise an alert every time one or more queries are activated.

As mentioned, this module acts as a service that generates alerts for each registered application in the system. These alerts need to be subscribed so the application will receive them. Later in this chapter we shall discuss the flow of the subscription to the alerts service.

5.2 Alerts Module Architecture

This subsection describes how the alerts are subscribed and generated. An overall architecture of the alerts module will be presented, as well as a simple use case of the system flow. For better understanding we have decided to grey out the least important parts in the overall architecture leaving the components used by this module in colour, this should provide a better understanding in terms of how everything comes together in the end. Additionally, we have added all the intermediary modules that were too specific to include in the high level architecture, with the intent of providing a more technical approach to this chapter as it documents the implementation of the alerts module. Figure 19 illustrates the general architecture of the system which includes the alerts module.

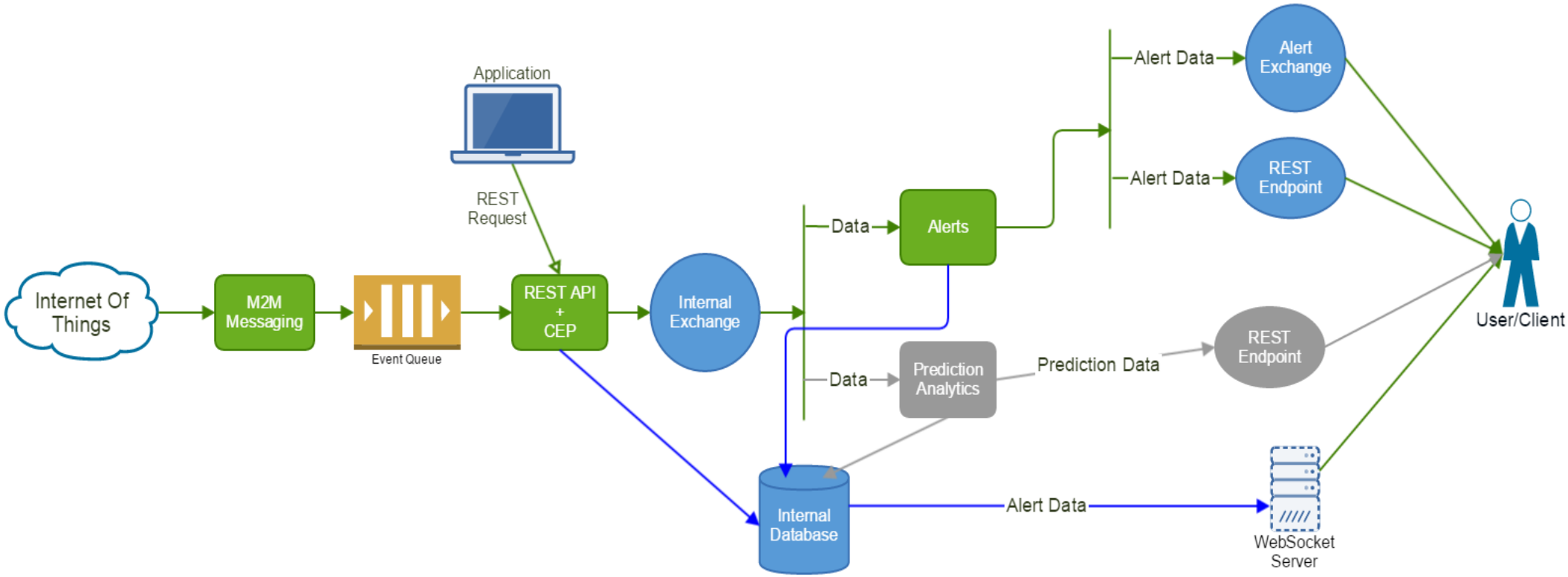

Figure 19 - Alerts module architecture

The alerts module intends to generate alerts in case something is not right in the incoming data. Furthermore we shall explain the role of each component, providing a deeper and more technical approach. The components are:

- Internet of Things – This component represents all the hardware layer that provides the data;
- M2M Messaging – This component is responsible for bridging the incoming data from the hardware layer to a meaningful and standardized event. In this component a technology that accepts multiple M2M protocols is of extreme importance, therefore we have chosen to integrate Meshblu (Meshblu, 2015), which is a M2M communication tool that contains these characteristics;
- Event Queue – This component intends to queue the incoming events to provide scalability and fault tolerance to the system. For this we have used RabbitMQ (RabbitMQ, 2015), which is one of the most popular messaging systems;
- REST API + CEP – This component is responsible for handling all the API calls, exposing REST endpoints and starting Esper engines that will receive and process the incoming data independently, for each subscribed application. Furthermore this component is also designed to persist the data that passes on the system. This data varies from application subscriptions, which can be made by sending a POST to the API, to the events coming from the hardware layer that need to be persisted for further processing by the analytics module;
- Application – This is an external component to the system, although it is important to include it in the overall architecture so that everyone can acknowledge how the system is started. Any application that wants to subscribe the system needs to do a POST in the provided API to become registered in the system. After that a configuration ID is generated and assigned to the requesting application;
- Internal Exchange – This component is responsible for the internal messaging between modules, metaphorically speaking it can be treated as a “modules bridge”. It intends to guarantee that all the components receive the alerts that are being generated by the Esper engine. Notice that the component is an exchange rather than a queue, this is supported by the fact that to each exchange multiple

queues can be added, these queues will then act as "listeners" to the internal modules that need to receive alerts;

- Internal Database – This component is responsible for storing all of the data. We have chosen to use Cassandra for its great abilities when writing, without sacrificing reading performance. Moreover we have conducted experiments, as described in chapter 4;
- Alerts – This is the module which handles the generated alerts by the Esper engine and delivers it to the users via AMQP, REST, WebSockets, WebHooks and Meshblu which intends to provide M2M communication. It is arguable that this module didn't need to be separated from the Esper engine, which allowed for the inexistence of messaging overhead between the two modules. Although from a scalability standpoint, being separated allows for a better and more controlled growth. Also, as the user can subscribe this module with many technologies (e.g. email, WebSockets, WebHooks, AMQP, amongst others), different paradigms are present. Let's assume that the user subscribed the alerts module and wants to receive data via AMQP and WebHooks, when an alert was generated it would be seamlessly published to the queue due to its non-blocking nature, on the other hand WebHooks are a blocking technology, which means that while the message is not delivered the system cannot resume its current operation. Thus, we have decided to implement a new module which is created from scratch for each application, which means that each application will have its own Alerts module. At runtime the Alerts module connects to the Internal Exchange via AMQP to receive the alerts that are being generated, then it publishes the alerts for that application via the registered technologies. As mentioned before there are different technologies at play in this module to overcome this, a thread is created and becomes responsible for sending alerts to the destination. In practical terms, this means that if the user has requested the alerts to be sent with AMQP and WebHooks, each of these technologies will have a thread specific to them responsible for sending the alerts;
- WebSocket Server – This component intends to provide WebSocket access to the alerts raised by the application. Although this intends to provide access to the alerts, it is separated from the module because it would be more complex to implement alongside the other technologies. A WebSocket server needs to have its

own message interpretation logic, it needs to expose endpoints for its clients to connect, to handle events and to be lightweight and real time. This was the main reason to separate the WebSocket server from the Alerts module;

- Alert Exchange – This component intends to provide AMQP access to the alerts. Once again an exchange was implemented for easier more dynamic connection, in other words if the user needs to have the alerts being sent to two distinct applications it can simple connect two queues to the exchange;
- REST Endpoint – This component is not a module by itself, it only represents the exposure of a REST endpoint for the alerts to be received via API calls.

Figure 20 shows the lifecycle for an event to be considered an alert of any type. The flow starts by receiving the event to which are then applied the rules defined by the user. For this specific use case, we have decided to implement only three rules which are illustrated by the three decision blocks in the diagram. These three rules intend to fire an alert if: (1) a user defined time is reached, this will result in a monitor alert; (2) a user defined threshold is reached, which results in a warning alert; (3) a user defined set of events is reached, which results in a critical alert. Note that these rules are provided by the application which is registered in the system in a format which is similar to SQL, but specific to the Esper engine that is EQL (Esper Query Language) (Esper, 2015).

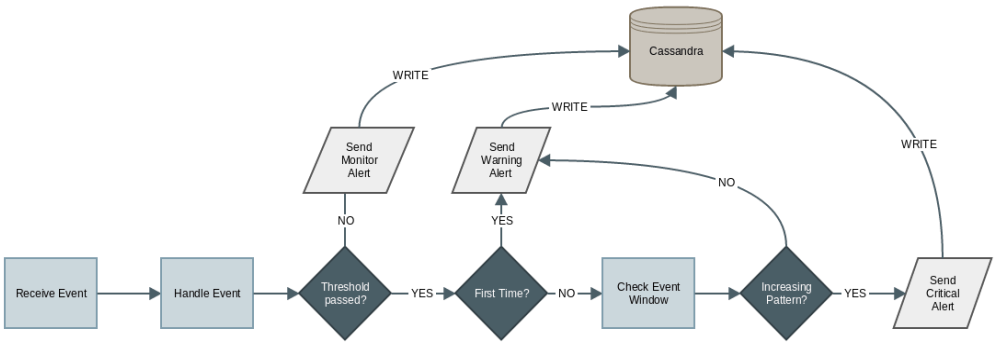

Figure 20 - Alerts module event flow

5.3 API endpoints

An endpoint is the entry point to a service or a process, usually seen on REST APIs.
This subsection lists the endpoints that are available via the API. For a clear understanding we have decided to do a list with the endpoints, explaining each one of them:

- POST /api/configuration – This endpoint is used to register a new configuration. It receives a JSON POST, with the configuration details. This endpoint will then

respond with a token (configuration_id) that needs to be used when accessing other endpoints on the API;

- POST /api/business_rule – This endpoint receives a JSON POST request. It is used to store the rules that each registered application needs to have. It automatically starts a listener on the queue that was registered in the endpoint configuration;
- GET /api/rules/{configuration_id} – This endpoint allows the application to retrieve the rules that have been registered, it is useful for a better management of each application rules;
- GET /api/alerts/{configuration_id} – This endpoint is used to retrieve the alerts that each application fires, it is useful if managers want to list the alerts that have been raised during the system deployment;
- POST /api/alert/access_method – This endpoint intends for the application to register with which technologies it wants to access the alerts.

5.4 How to receive alerts?

This subsection intends to explain how it is possible for an application to register in the system and start receiving alerts. Figure 21 illustrates the necessary steps to start receiving alerts.

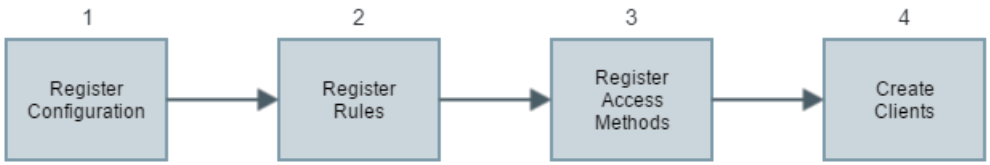

Figure 21 - Steps to receive alerts

As we can see in Figure 21 the flow is very simple:

1. Register a configuration that will have the source of the incoming real time data;
2. Register the rules that need to be listening in the data for patterns. If a configuration already exists queries can be added to it, without needing to create a different configuration;
3. Register the desired access methods to receive the alerts generated by the supplied rules. More access methods can be added after rules are registered (e.g. WebSockets, AMQP);
4. Create one or more clients that will receive and process the alerts that are being generated.

5.5 Summary

This module is the central part of the system, providing a Complex Event Processing engine to generate alerts based on user defined criteria. It is a very important piece of the Citibrain platform because it provides a unified control centre responsible for monitoring every other application of the platform. Beyond generating alerts, this module provides an elegant, seamless and easy way to get access to the alerts via different technologies that will empower any system. These applications range from a simple email that can be sent to a technician, to a more complex M2M system that can warn other machines when an alert is generated and initiate an emergency protocol.

In short, the Alerts Module is responsible for generating alerts when anything is out of the ordinary and guarantee that these alerts are sent to people so they can act on the emergency.

6. PREDICTIVE ANALYTICS MODULE

This chapter presents information on the predictive analytics module, which will be implemented with the purpose of providing useful knowledge to emergency response teams.

For the predictive analysis module it was decided that the PredictionIO (PredictionIO, 2015) machine learning server would be the chosen because it is open-source and provides a good platform, also it uses Apache Spark as its main machine learning data processing engine.

This module intends to empower the system by providing:

- Trends which inform experts on indicators tendency (e.g. temperature rising, water level rising);
- Predictions which allow for better prepared emergency teams.

6.1 Predictive Analytics Overview

Predictive Analytics (Finlay, 2014) is a business intelligence technology that aims to predict a result for each business case. In other words, it is the process of extracting information from datasets with the intention to discover hidden patterns and predict future outcomes and trends. It is important to understand that predictive analytics does not predict the future, it only estimates what might happen in the future with an associated degree of confidence.

Furthermore, according to (Finlay, 2014) Predictive Analytics "is not new", it is stated that the earliest applications were credit scoring in the 1950s, which became, by the mid-1980s, the main decision-making tool in financial services. A good example of how Predictive Analytics work is in the banking business. When a person asks for a loan in a bank, the bank manager will test the characteristics of this person against old data, which will predict if the loan will be successfully paid, or not. Although this type of analyses is very useful, most of the times it needs input from experts in the field, thus these predictions must always be analysed by domain experts.

The main idea behind this type of analysis is to forecast future events based on past data. Thus, it is necessary to use the past data to build a predictive model, which will be used to test the new events.

PredictionIO (Chan et al., 2014; PredictionIO, 2015) was the tool chosen to do the prediction module. PredictionIO is an open-source machine learning server with the ability to deploy predictive analytics in a short period of time, built on top of Apache Spark. According to PredictionIO it "eliminates the friction between software development, data science and production deployment" (PredictionIO, 2015).

Furthermore, according to PredictionIO consists on the following components:

- PredictionIO platform, which is composed by all the tools for building, evaluating and deploying machine learning engines. Figure 22 shows PredictionIO's high level architecture.

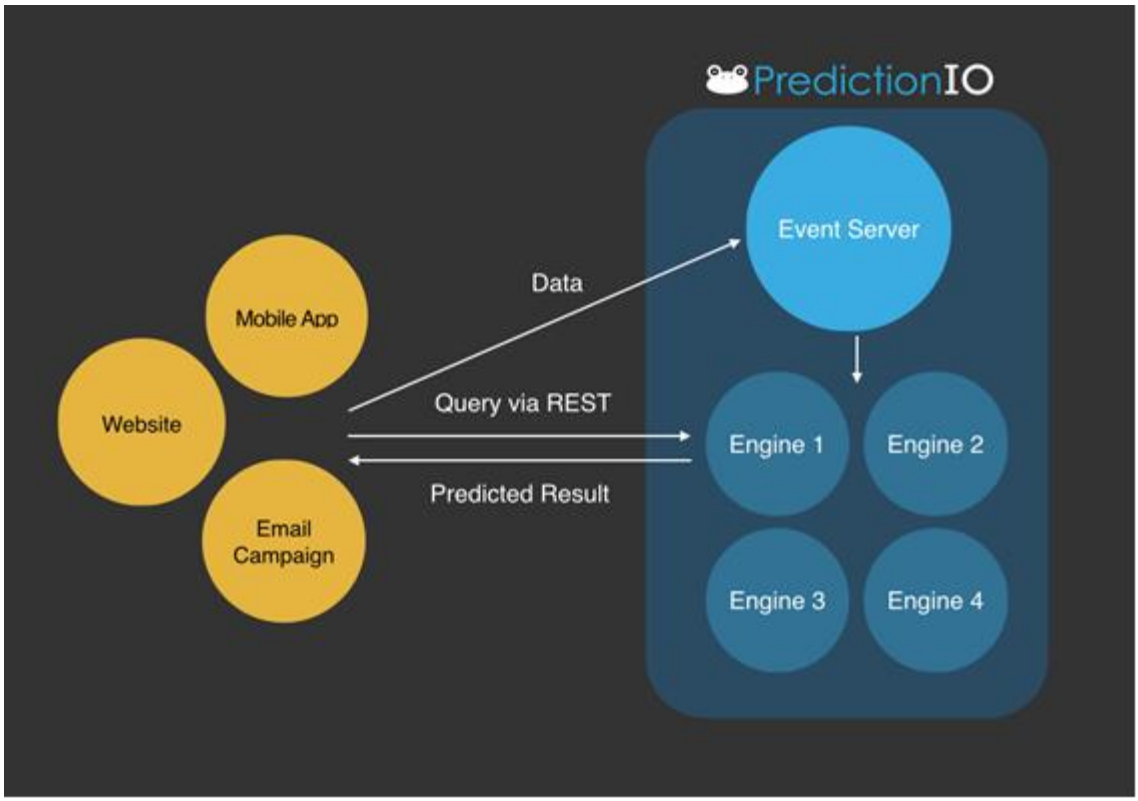

Figure 22 - PredictionIO high level architecture (PredictionIO, 2015)

It is important to note that this architecture has a multiple engine configuration, which means that each engine is independent from the others and can be assigned to a specific task, such as clustering or classification;

- Event Server collects the data from the user's application. The engine will then build the model using the chosen algorithm. Figure 23 gives a very good perspective of this module.

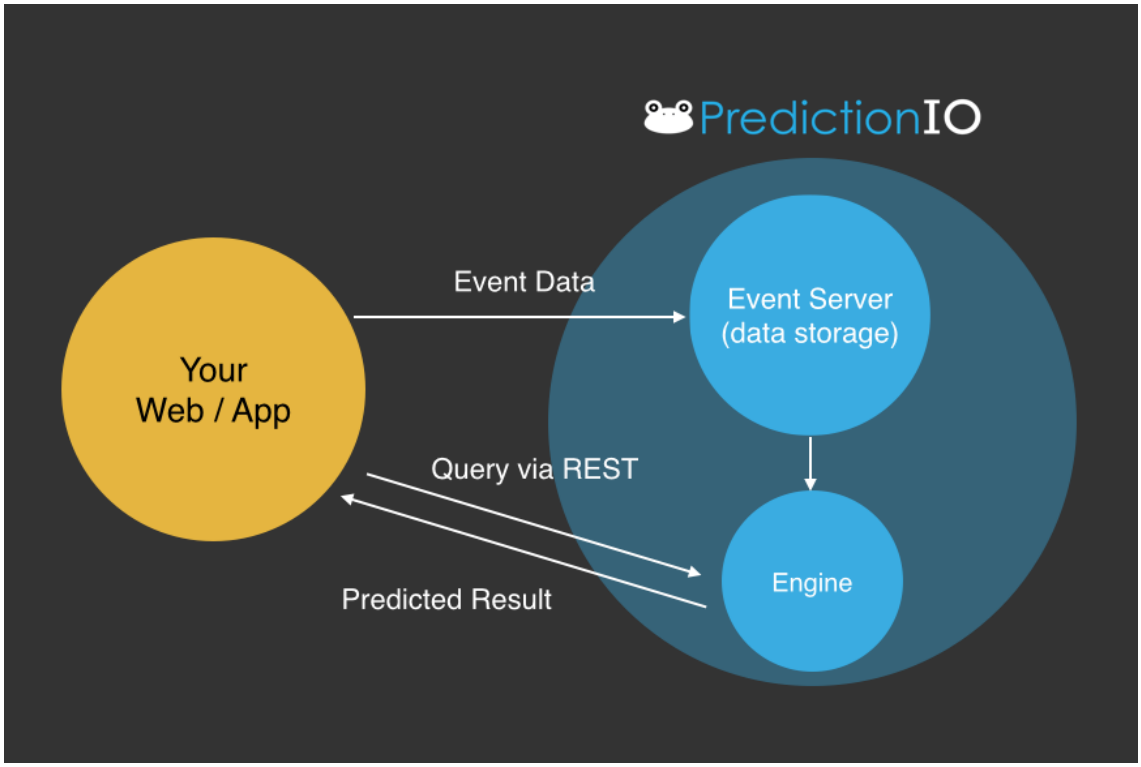

Figure 23 - PredictionIO event server architecture (PredictionIO, 2015)

In contrast to Figure 22 where there was a multiple engine configuration, in Figure 23 there is only one engine which is assigned to do a specific machine learning task (e.g. running a prediction algorithm to forecast the temperature in a specific place) in the requesting application.

6.2 Predictive Analytics Module Architecture

In this subsection we show how the predictions are made. An overall architecture of the predictive analytics module will be presented.

The module architecture will be described using the same approach as in the alerts module chapter. For better understanding we have decided to grey out the least important parts in the overall architecture leaving the components used by this module in colour, this should provide a better understanding in terms of how everything comes together in the end. Additionally, we have added all the intermediary modules that were too specific to include in the high level architecture, the intent is to provide a more technical approach to this chapter as it documents the implementation of the predictive analytics module.

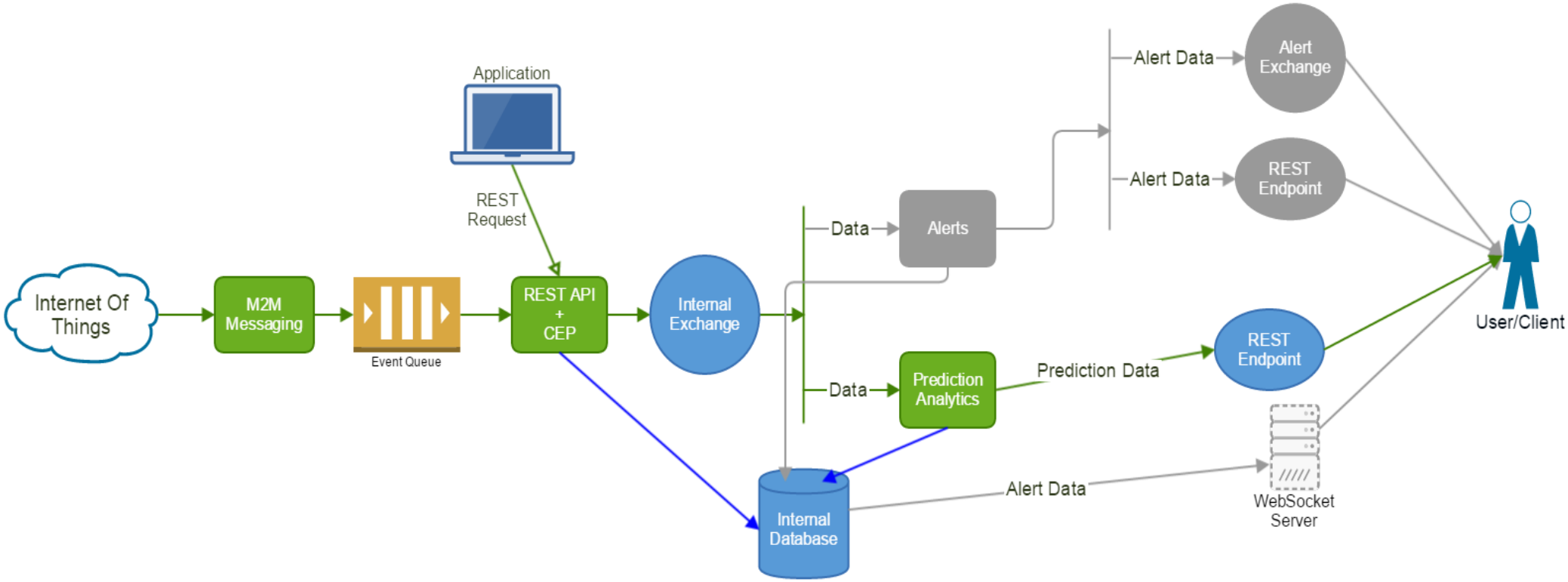

Figure 24 - Prediction module architecture

In Figure 24 we can observe the system's overall architecture although, for this case, we shall focus on the predictive analytics module, which allows users to receive predictions based on past data. The explanation of the modules in this chapter will not be as extensive as in the alerts module, because the general internal processes of each of them is the same. Thus, we shall explain the two new modules that have been added:

- Prediction Analytics – this module is responsible for making predictions based on the dataset provided by the user. These predictions can be for different types of events, such as environment, traffic, water, amongst others. The intention behind this module is to allow the users to be better prepared for a future problem. This module will consume the data from the internal exchange to analyse with a prediction algorithm. The algorithm will analyse the request made by the user and estimate based on past data, which is the most probable scenario for that specific case. The generated prediction is of extreme importance for emergency response teams as it can actively estimate when a disaster is more likely to occur;
- REST Endpoint – the REST endpoint serves as a way for the user to get the predictions from the application, these predictions are pre computed by the algorithm.

6.3 Dataset

The dataset used is from sensors that are already in place in a city. The intention behind using real data is to approximate the model that will be trained to a real scenario, providing a better approach to what we are trying to achieve.

The data provided in the dataset comes from environmental sensors. It measures units such as temperature, precipitation, noise level, carbon dioxide, amongst others.

The data on this dataset is not pre-processed, which means some processing might be needed, because it might contain empty values, or outliers which might reflect errors on the sensor, or the sensor reading. To overcome this, a clustering algorithm will be used, the intention is to label the data. Clustering was chosen because it is a very relevant technique when dealing with unlabelled data. It creates clusters which aggregate similar items, these clusters are then assigned with a label which will later serve to classify each new incoming item.

After being pre-processed and labelled, data will act as an input for the algorithm being used at that time. As an example, regression can be used on this case. Regression is useful

for predicting a future value because it is based on a statistical method that estimates relationships among variables. This is advantageous because, from past data, the algorithm is able to estimate which is the most probable value for a given point.

For a better understanding an example shall be presented: let's assume it is winter and we have a garden which, for the sake of this example, only receives water when it rains and we want to predict how fast the grass will grow. Based on previous data we know that during the winter it rains more, therefore the grass should grow a lot faster. The algorithm analyses the past data and encounters a similar case and for it the grass has grown 0.3 of its original size. From this we can assume that the grass will grow about the same.

From a mathematical standpoint linear regression analysis is nothing more than an association between two variables, which can be translated by the equation of a straight line, $y = mx + b$ where y is the value we want to predict, m is the line slope, x is the value from which we want to predict and b the interception of the y axis.

Furthermore, an implementation overview shall be made to clarify some aspects of the tools used.

6.4 Predictive Analytics Module Implementation

From an implementation standpoint it was decided to use clustering for the classification of the dataset, which allows for the classification of unlabelled data. This provides labels to the data which will be very important when applying other modules. These other modules serve to classify each new incoming event and were trained with the labelled dataset from the clustering output. Labelling the dataset allows for the usage of supervised learning algorithms, which only work on labelled datasets.

Furthermore we shall present algorithms that can be used for each step. Keep in mind that, due to this part of the system is still in development phase, these are not necessarily the choices that will be made, but are great contenders to be included:

- Clustering – k-means, DBSCAN;
- Classification – Support Vector Machines, Neural Networks;
- Prediction – Regression, Gradient Boosting Machines, Random Forests.

As these algorithms were designed for different tasks, the implementation needs to be made on different phases and from different perspectives. For instance clustering is only needed when the dataset is unlabelled, with the intention of labelling it. On the other hand, classification will be a recurrent task for each incoming event. Regression can be

used iteratively, or at each request, depends on user's interaction. It is also important to note that the predicted value can be accessed via the technologies mentioned in the alerts chapter.

At this point the system is still in development phase, therefore results are not available. Although from this module it is expected to receive results such as:

- Trend indicators for the temporal analysis of data, which allow for a better decision making process;
- Predictions on future events (e.g. fire, floods, traffic congestions);
- The most affected areas by certain event (e.g. which city area is more likely to have a traffic congestion);
- The most affected periods of time by event (e.g. which time of the day is more likely to have a traffic congestion).

These results will provide the necessary knowledge to boost the system's overall capabilities, providing a way of estimating future events.

7. CASE STUDY: URBAN SAFETY SYSTEM WITHIN OPORTO'S CITIBRAIN NODE

In this chapter we aim to provide a use case to describe the usage of the prototype. Before the development of this project, Citibrain lacked a solution that could monitor the data which was being produced by its applications. With this in mind, a system that could monitor city parameters such as temperature, humidity, ozone, amongst others was considered. The project aimed to develop a prototype that could generate alerts based on thresholds provided by users. These thresholds come in the form of a rule which generates the alerts.

Citibrain is a consortium specialized in smart solutions, which aims to create desirable and liveable places, bringing together cities and citizens to improve the quality of life through technology. This is achieved with the implementation of applications in different sectors. Figure 25 shows the current applications offered by the Citibrain platform.

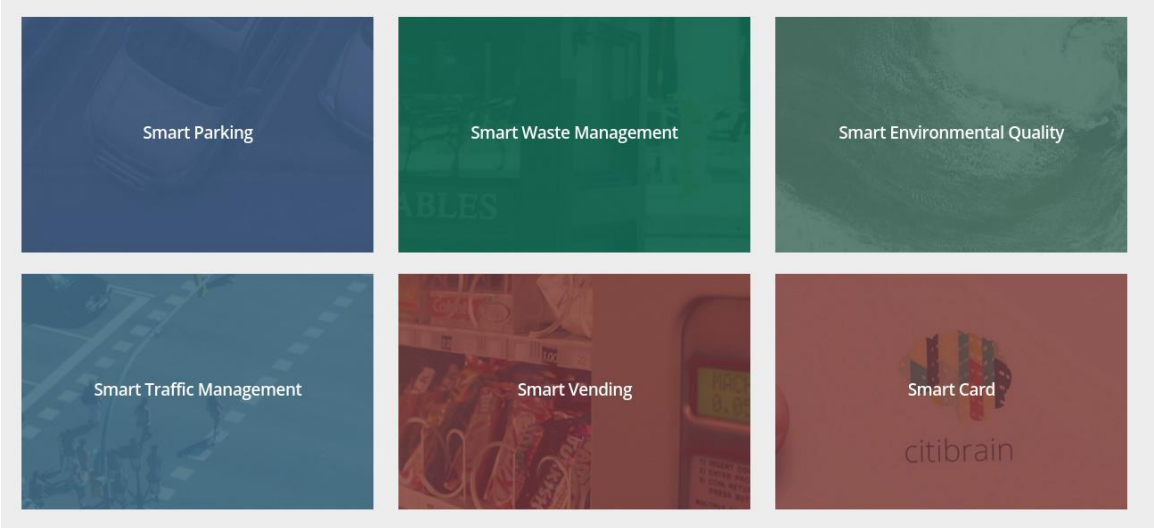

Figure 25 - Citibrain Applications

This case study is focused on the Smart Environmental Quality application. This application uses small sensing stations installed in the current urban infrastructure, making possible the collection of indicators on air quality, noise pollution levels, temperature, atmospheric pressure, humidity and luminosity (Citibrain, 2015).

The Oporto city was chosen because it was one of the first cities to have sensors and to provide data to the Citibrain platform. Oporto's network of environmental sensors provides real time data of the environmental status of the city (e.g. noise levels,

temperature readings, ozone levels, carbon dioxide levels). Figure 26 shows two hours of temperature data from a sensor in Oporto. This data comes from a real sensor deployed on Oporto and is feeding the Citibrain platform. The bars do not represent the temperature values, they represent the number of gathered temperature events at every five minutes.

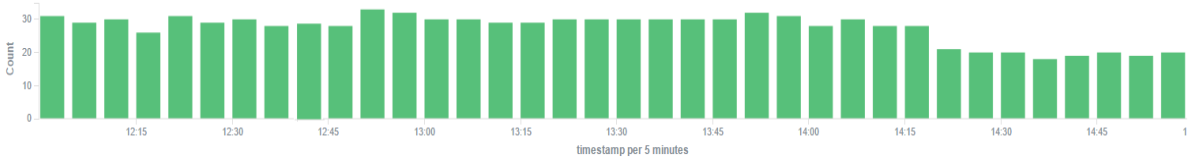

Figure 26 - Temperature data

With the developed prototype it is now possible to analyse the data stream and generate alerts when anything abnormal is happening. For example, with the temperature data it is possible to alert entities of a sudden temperature increase, if the value passes the defined threshold. This could help prevent dangers related to the temperature (e.g. fire). These alerts are very important as they provide a feasible way to be in control of the indicators that come from the sensors around the city. Also, entities responsible for the city safety will be more prepared to act in case of any problems.

The system is ready to receive multiple streams of data, which provides a way of preventing dangers with more accuracy.

A temperature sensor alone is not an exact way of preventing a fire, because it can send data which is not related to a fire (e.g. direct sunlight could have made the temperature rise). With the addition of a smoke sensor the system can provide more accurate results because it contains two sources of information.

Another feature that the prototype brings to the platform is the capability of predicting future events based on past data. In other words, the platform is capable of analysing historical data and deliver conclusions from it. For example, the prototype is able to predict if a certain set of events will result in a disaster (e.g. temperature rising resulting in a fire). These predictions allow for the responsible entities to plan ahead and take measures to avoid disasters, also predictions will make possible to manage and organize emergency teams in a more efficient way.

The system has a control centre which will aid users when looking for a convenient, elegant way of managing their data. Figure 27 shows the dashboard for the Citibrain control centre. The control sensor contains information from all the applications which are being used by the user. The information is shown on cards which are customizable, this means that the user can drag and drop the wanted cards to enhance the overall view

of the data. In Figure 27 we can see four cards: on the top left, we can see a map with parking spots, this data comes from the Smart Parking application; on the top right we can see the live water pressure for a pipe, this data comes from the Smart Water application; on the bottom left we see live water pressure from another pipe; on the bottom right we can observe waste containers on a map. This data comes from the Smart Waste application.

Another important aspect of Figure 27 is the menu on the left side, which helps for quick and easy access to:

- Events – This menu shows alerts in real time. The developed prototype will be integrated in this module, providing a way for users to see their alerts in real time.
- Dashboards – This menu shows the custom dashboards created by the user. It helps for quick access all the dashboards;
- Management – This menu gives the user the possibility to manage the general settings of the dashboard;
- Apps – This menu serves as a quick way to go to any subscribed application and see what is happening in real time.

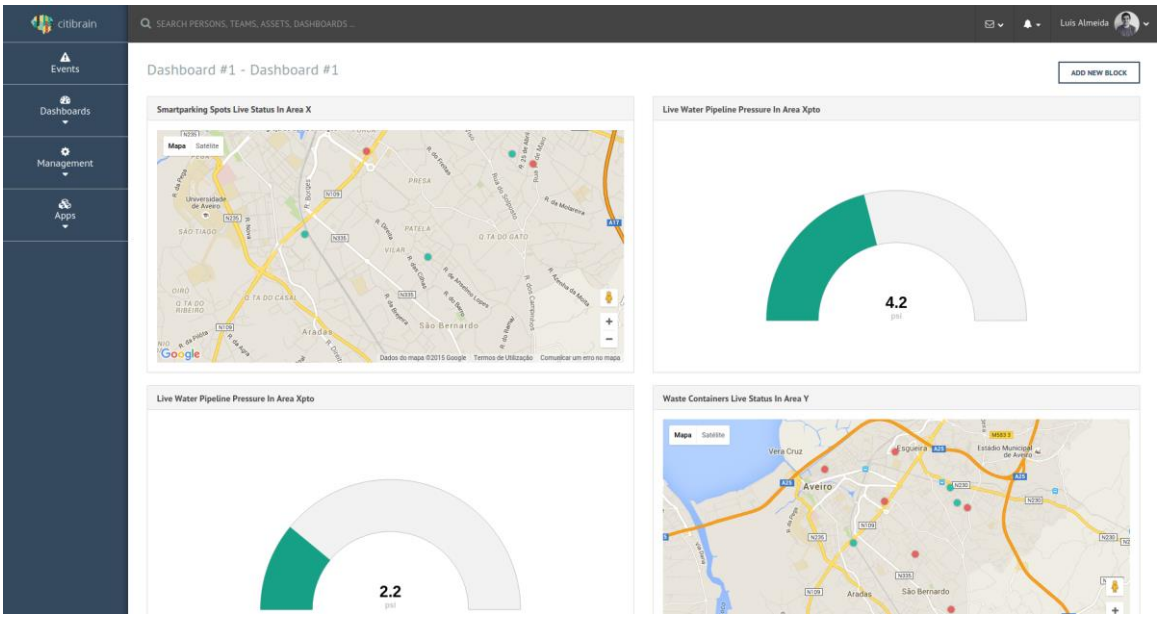

Figure 27 - Citibrain dashboard

This control centre will also act as a manager for the applications of that user. Figure 28 shows a Smart Environment application which is receiving data from sensors. The list in Figure 28 is composed by data sent by the sensors to the application. The list is filled by data from different types of sensors. In this example we can see data from:

- Temperature Sensors – These measure the temperature on a specific place;

- Precipitation Sensors – These measure the amount of precipitation on a specific place;
- Wind Speed Sensors – These measure the velocity of the wind for a specific site.

Furthermore the list also contains values from processed data, this means that these indicators were already processed by algorithms that analyse if the value is within the acceptable thresholds or not.

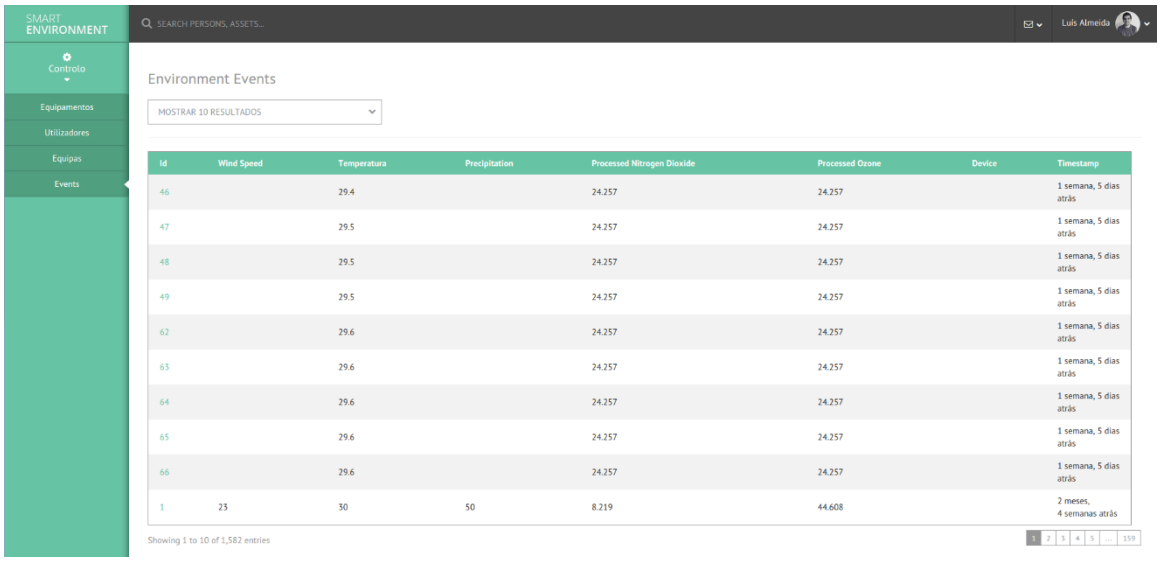

Figure 28 - Smart Environment events dashboard

This case study helped to understand the importance of the alerts module in the Citibrain platform. The module is important to capture the data coming from streams and generate alerts to make the entities responsible for city safety more aware of incoming dangers. The module is a central piece in the Citibrain platform and will be integrated as a core functionality. The integration with the rest of the platform will be seamless, this means that the module is prepared for handling data from every subscribed application providing an elegant and effortless way of monitoring city problems.

8. CONCLUSIONS AND FUTURE WORK

The demands placed on the city ecosystem by the population are increasing. This directly affects the entities responsible for the city safety and control. The existing systems do not provide the necessary tools to engage with city surroundings, which can difficult the decision making process of city leaders.

This work aimed to develop a system which would monitor city parameters such as temperature, humidity, ozone, amongst others. The main goal of the system is to generate alerts based on thresholds provided by users, which can raise their awareness of a forthcoming danger.

The implementation of this system made possible the accomplishment of the proposed main contributions:

- An Application Program Interface (API), which intends to gather data from sensors scattered around the city;
- An engine capable of receiving, processing and dispatching the alerts to emergency personnel;
- We have shown a way to develop a smart system which receives streams of data from many sources and delivers knowledge;
- The presentation of an architecture for the collection of data in the IoT;
- The creation of a system which generates alerts to inform experts and enhance the overall city safety.

After the internship, the prototype will be tested and integrated in the Citibrain platform as one of its components. It will add new capabilities to the platform such as the generation of alerts and ways of estimating future events that are likely to happen.

On a more practical note, the internship made possible the development of a M2M prototype which acts as an emergency and security controller. The main outputs of the internship are an API which intends to register applications which retrieve data from the sensors and an engine capable of handling, processing and dispatching the alerts to emergency personnel.

On a personal and professional level, the student considers the internship to be a rich experience. The choice for an auto proposed internship on a company was made because

it is an essential complement to the degree since it allows learning in a professional environment providing a better, more prepared future employee.

To the intern the result of this internship was very positive. It was given the possibility of integration in a business environment where there were real situations which need to be given rapid responses. Also, working with qualified professionals with extensive knowledge allowed me to acquire some of extra skills and work customs.

From a practical standpoint, during the internship, the tasks that where presented allowed me to enhance technical skills, as well as the learning of new ones. The knowledge that was shared was rewarding and provided motivation for a future entrance in the professional world.

In short the internship, allowed the learning of new skills at a technical and personal level, as well as the application of already acquired skills during the degree. The proposed solution appears to meet the expectations of both the intern and the company. The difficulties related to the new technologies and the new work methodology have been overwhelmed and provided a more prepared vision for the future in the area.

Generally speaking a software is never ended and has always something more to add, this is also the case with the developed system. As future work we aim to provide some notions of how the system can evolve and which features we think are the best for future implementation.

An interesting addition to be developed in the future, is the inclusion of social mining. Due to the importance of social networking in nowadays society seems like and excellent way to complement the inputs of the system. This is important to complement the system because it can detect disasters via a post in a social network. The post does not need to be in a specific format, the algorithms will only be looking for keywords that will trigger the attention of the system. Although this data is extremely relevant, it is important to guarantee that it isn't false. A possible solution for this problem can be a request to the sensors that are placed in that specific site.

Another interesting module to add to the system would be outlier detection. Outliers are observation points that are distant from the rest of the dataset. This would provide a way to detect whether the sensors are calibrated or stopped sending the information feed. This is only possible because an event of 'null' would be considered a distant value from the average. In the overall architecture, the outlier detection module could be inserted

immediately after the event queue sends the events for processing, this would allow the outlier detection module to send an alert in case of any problems with the hardware.

In short, many more features could be added to a complex system such as this one, we have decided to point out these two, because they might be the most important to implement in a near future.

REFERENCES

Abramova, V. and Bernardino, J. (2013). NoSQL databases. Proceedings of the International C* Conference on Computer Science and Software Engineering - C3S2E '13.

Abramova, V., Bernardino, J. and Furtado, P. (2014a). Evaluating Cassandra Scalability with YCSB. Database and Expert Systems Applications, pp.199-207.

Abramova, V., Bernardino, J. and Furtado, P. (2014b). Which NoSQL Database? A Performance Overview. Open Journal of Databases (OJDB), Volume 1(Issue 2).

Abramova, V., Bernardino, J. and Furtado, P. (2014c). Testing Cloud Benchmark Scalability with Cassandra. 2014 IEEE World Congress on Services.

Albtoush R., Dobrescu R., Ionescou F., A Hierarchical Model for Emergency Management Systems, 2011.

Ankit Toshniwal, Siddarth Taneja, Amit Shukla, Karthik Ramasamy, Jignesh M. Patel, Sanjeev Kulkarni, Jason Jackson, Krishna Gade, Maosong Fu, Jake Donham, Nikunj Bhagat, Sailesh Mittal, and Dmitriy Ryaboy. Storm@twitter. In Proceedings of the 2014 ACM SIGMOD international conference on Management of data (SIGMOD '14) 2014

Anttiroiko, A., Valkama, P. and Bailey, S. J., (2014). Smart cities in the new service economy: building platforms for smart services. AI Soc. 29(3): 323-334

Apache Hadoop - http://hadoop.apache.org/ [Accessed 5 Aug. 2015]

Apache Storm - http://storm.apache.org/ [Accessed 5 Aug. 2015]

Athena Vakali, Leonidas Anthopoulos, and Srdjan Krco. Smart Cities Data Streams Integration: experimenting with Internet of Things and social data flows. In Proceedings of the 4th International Conference on Web Intelligence, Mining and Semantics (WIMS14) 2014.

Aurigi, A. (2005) Making the digital city. The early shaping of urban internet space. Ashgate, Aldershot

Benkhelifa, I.; Nouali-Taboudjemat, N.; Moussaoui, S., "Disaster Management Projects Using Wireless Sensor Networks: An Overview," Advanced Information Networking and Applications Workshops (WAINA), 2014 28th International Conference on, vol., no., pp.605, 610, 13-16 May 2014.

Carillo FJ (ed) (2006) Knowledge cities. Approaches, experiences, and perspectives. Elsevier, Amsterdam

Cecchinel, C.; Jimenez, M.; Mosser, S.; Riveill, M., "An Architecture to Support the Collection of Big Data in the Internet of Things," Services (SERVICES), 2014 IEEE World Congress on , vol., no., pp.442,449, June 27 2014-July 2 2014.

Charsyam - Cassandra Data Model - https://charsyam.wordpress.com/tag/cassandra-data-model/ [online] Available at: [Accessed 08-01-2015].

Ching Yu Chen; Jui Hsi Fu; Sung, T.; Ping-Feng Wang; Jou, E.; Ming-Whei Feng, "Complex event processing for the Internet of Things and its applications," Automation Science and Engineering (CASE), 2014 IEEE International Conference on , vol., no., pp.1144,1149, 18-22 Aug. 2014.

Chourabi, H.; Taewoo Nam; Walker, S.; Gil-Garcia, J.R.; Mellouli, S.; Nahon, Karine; Pardo, T.A.; Scholl, Hans Jochen, "Understanding Smart Cities: An Integrative Framework," System Science (HICSS), 2012 45th Hawaii International Conference on , vol., no., pp.2289,2297, 4-7 Jan. 2012

Citibrain.com, (2015). Citibrain. [online] Available at: http://citibrain.com [Accessed 10 Aug. 2015].

DataStax, (2014). ALLOW FILTERING explained. [online] Available at: http://www.datastax.com/dev/blog/allow-filtering-explained-2 [Accessed 5 Jul. 2015].

DB-Engines Ranking [online] http://db-engines.com/en/ranking [Accessed 22 April of 2015]

Description of implemented IoT services – http://smartsantander.eu/downloads/Deliverables/D4.2.pdf.

Docs.datastax.com, (2015). Apache Cassandra™ 2.0. [online] Available at: http://docs.datastax.com/en/cassandra/2.0/cassandra/architecture/architectureDataDistributeReplication_c.html [Accessed 25 Oct. 2015].

Esper-epl-tryout.appspot.com, (2015). EsperTech - Esper Enterprise Edition: Enterprise ready Event Processing and CEP platform. [online] Available at: http://esper-epl-tryout.appspot.com/epltryout/index.html [Accessed 15 Aug. 2015].

Espertech.com, (2015). Chapter 5. EPL Reference: Clauses. [online] Available at: http://www.espertech.com/esper/release-5.2.0/esper-reference/html/epl_clauses.html#epl-intro [Accessed 5 Aug. 2015].

Espertech.com, (2015). EsperTech - Products - Esper. [online] Available at: http://www.espertech.com/products/esper.php [Accessed 5 Aug. 2015].

European Network of Living Labs, (2014). [online] Available at: http://www.openlivinglabs.eu/ [Accessed 5 Aug. 2015]

Finlay, S. (2014). Predictive analytics, data mining and big data. Palgrave Macmillan.

Forum Virium Helsinki, (2014). Available at: http://forumvirium.fi/en [Accessed 5 Aug. 2015]

Google Developers, (2015). Chrome V8 | Google Developers. [online]. Available at: https://developers.google.com/v8/ [Accessed 5 Sep. 2015].

Hewitt, E. (2011). Cassandra The definitive guide. Beijing. O'Reilly.

Hollands RG (2008) Will the real smart city please stand up? Intelligent, progressive or entrepreneurial? City 12(3):303–320

Höller, J.,,Tsiatsis, V., Mulligan, C., Karnouskos, S. Avesand, S. and Boyle D.,, From Machine-to-Machine to the Internet of Things: Introduction to a New Age of Intelligence. Amsterdam, The Netherlands: Elsevier, 2014.

http://amsterdamsmartcity.com/?lang=en [Accessed 5 Aug. 2015]

http://opencities.net/barcelona [Accessed 5 Aug. 2015]

http://www.smartsantander.eu/index.php/testbeds/item/132-santander-summary [Accessed 5 Aug. 2015]

Ics.uci.edu, (2015). Fielding Dissertation: CHAPTER 5: Representational State Transfer (REST). [online] Available at: https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm [Accessed 27 Aug. 2015].

Islam, S.M.; Kwak D., Kabir H., Hossain, M., Kyung-Sup Kwak, "The Internet of Things for Health Care: A Comprehensive Survey," in Access, IEEE , vol.3, no., pp.678-708, 2015

Itria, M. L., Daidone, A., Resiltech, S. R. L., & Ceccarelli, A. A Complex Event Processing Approach for Crisis- Management Systems Exploiting Crowd Sourcing and Crowd Sensing Information for Situational Awareness. 2014.

Jara, A.J.; Genoud, D.; Bocchi, Y., "Big Data in Smart Cities: From Poisson to Human Dynamics," Advanced Information Networking and Applications Workshops (WAINA), 2014 28th International Conference on , vol., no., pp.785,790, 13-16 May 2014

Jiafu Wan; Di Li; Caifeng Zou; Keliang Zhou, "M2M Communications for Smart City: An Event-Based Architecture," Computer and Information Technology (CIT), 2012 IEEE 12th International Conference on , vol., no., pp.895,900, 27-29 Oct. 2012.

Joann J. Ordille, Patrick Tendick, and Qian Yang. 2009. Publish-subscribe services for urgent and emergency response. In Proceedings of the Fourth International ICST Conference on Communication System software and middleware (COMSWARE '09), 2009.

Komninos N (2002) Intelligent cities. Innovation knowledge systems and digital spaces. Spon Press, London

Lakshman, A. and Malik, P. (2010). Cassandra. SIGOPS Oper. Syst. Rev., 44(2), p.35.

Lambda Architecture - http://lambda-architecture.net/ [Accessed 5 Aug. 2015]

Lei Gu; Huan Li, "Memory or Time: Performance Evaluation for Iterative Operation on Hadoop and Spark," High Performance Computing and Communications & 2013 IEEE International Conference on Embedded and Ubiquitous Computing (HPCC_EUC), 2013 IEEE 10th International Conference on , vol., no., pp.721,727, 13-15 Nov. 2013.

Meshblu, (2015). Meshblu. [online] Available at: https://developer.octoblu.com/ [Accessed 5 Aug. 2015].

Nodejs.org, (2015). Node.js. [online] Available at: https://nodejs.org/ [Accessed 27 Aug. 2015].

Oracle.com, (2015). New to Java Programming Center - Downloads. [online] Available at: http://www.oracle.com/technetwork/topics/newtojava/downloads/index.html [Accessed 27 Aug. 2015].

P. Friess and O. Vermesan, Internet of Things: Converging Technologies for Smart Environments and Integrated Ecosystems. Aalborg, Denmark: River Publishers, 2013.

Piro G., Cianci I., Grieco L. A., Boggia G., and Camarda, P. 2014. Information centric services in Smart Cities. J. Syst. Softw. 88

PredictionIO - Open Source Machine Learning Server. [online] Available at: http://prediction.io [Accessed 25 Aug. 2015].

Rabbitmq.com, (2015). RabbitMQ - Messaging that just works. [online] Available at: https://www.rabbitmq.com/ [Accessed 5 Aug. 2015].

Radianti, J.; Gonzalez, J.J.; Granmo, O.-C., "Publish-subscribe smartphone sensing platform for the acute phase of a disaster: A framework for emergency management

support," Pervasive Computing and Communications Workshops (PERCOM Workshops), 2014 IEEE International Conference on, vol., no., pp.285, 290, 24-28 March 2014.

Simon Chan, Thomas Stone, Kit Pang Szeto, and Ka Hou Chan. 2013. PredictionIO: a distributed machine learning server for practical software development. In Proceedings of the 22nd ACM international conference on Conference on information & knowledge management (CIKM '13).

Socket.io, (2015). socket.io. [online] Available at: http://socket.io/ [Accessed 27 Aug. 2015].

Spark.apache.org, (2015). Overview - Spark 1.4.1 Documentation. [online] Available at: http://spark.apache.org/docs/latest/index.html [Accessed 5 Sep. 2015].

Spring.io, (2015). spring.io. [online] Available at: http://spring.io/ [Accessed 27 Aug. 2015].

Strickland, R. (2014). Cassandra high availability. Birmingham. Packt Publishing.

van der Veen, J.S.; van der Waaij, B.; Meijer, R.J., "Sensor Data Storage Performance: SQL or NoSQL, Physical or Virtual," Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on , vol., no., pp.431,438, 24-29 June 2012 doi: 10.1109/CLOUD.2012.18

Yi-Heng Feng; Lee, C.J., "Exploring Development of Service-Oriented Architecture for Next Generation Emergency Management System," Advanced Information Networking and Applications Workshops (WAINA), 2010 IEEE 24th International Conference on , vol., no., pp.557,561, 20-23 April 2010.

Zubaida Alazawi, Omar Alani, Mohmmad B. Abdljabar, Saleh Altowaijri, and Rashid Mehmood. 2014. A smart disaster management system for future cities. In Proceedings of the 2014 ACM international workshop on Wireless and mobile technologies for smart cities (WiMobCity '14), 2014.

ANNEXES

ANNEX A –
INTERNSHIP PROPOSAL

PROPOSTA DE ESTÁGIO

Ano Letivo de 2014/2015

em Mestrado em Informática e Sistemas (Business Intelligence)

TEMA

M2M Emergency System & Urban Safety

SUMÁRIO

Pretende-se desenvolver um sistema que monitorize parâmetros como temperatura, humidade, pressão atmosférica, radiação ultra-violeta, nível de ozono e CO2, detecção de incêndios e de inundações, em vários pontos de uma cidade em que o risco de os limiares de segurança serem ultrapassados seja elevado, e se crie então suporte para um sistema inteligente de apoio à decisão dos organismos de protecção civil.
Tendo este sistema informação em tempo quase real, podem observar-se tendências e fazer previsões (Data Analytics, Complex Event Processing, Inteligência Aritificial) para que sejam tomadas medidas preventivas ou reativas face à possibilidade de emergências ambientais ou civis. Estes sistemas de apoio à decisão podem então despoletar mecanismos de alarmística que por sua vez poderiam ser integrados com a restante infraestrutura da cidade, nomeadamente sinalização vertical, luminárias, sistemas de irrigação, sinalização de evacuamento, entre outras. Pretende-se ainda que este sistema esteja integrado com outros sistemas M2M em desenvolvimento na Ubiwhere, podendo partilhar dados com estes, de forma a possibilitar a gestão unificada de um cenário urbano.

O presente trabalho visa o desenvolvimento de um protótipo M2M para gestão de emergência e segurança em ambiente urbano. Pretende-se criar uma API para obter dados de sensores reais, dispersos através de uma cidade, bem como de API third-party e efectuar o tratamento e agregação dos mesmos para assim fornecer dados relevantes, em tempo-real, aos responsáveis pela segurança pública e protecção civil.
Pretende-se também investigar mecanismos de actuação, para que assim se possam criar acções automáticas que possam auxiliar na resolução das situações identificadas. Um exemplo destes mecanismos é a automatização da sinalização vertical para permitir a passagem de veículos de emergência com segurança até ao seu destino.

1. ÂMBITO

Contexto e justificação da validade do estágio proposto.

2. OBJECTIVOS

Pretende obter-se, no final da dissertação:

- Estudo do Estado da Arte em sistemas análogos
- Documento de Requisitos e Arquitectura da Aplicação Vertical M2M
- Módulo de Interoperabilidade com o Middleware M2M
- Camada de Integração com sensores em ambiente experimental
- Protótipo de User Interface simples para apresentação dos resultados

3. PROGRAMA DE TRABALHOS

O estágio consistirá nas seguintes atividades e respetivas tarefas:

- ***T1 – Elaboração do estudo do Estado da Arte***
- ***T2 – Levantamento e Especificação de Requisitos***
- ***T3 – Desenvolvimento da solução***
- ***T4 – Testes***
- ***T5 – Elaboração da Dissertação***

4. CALENDARIZAÇÃO DAS TAREFAS

As Tarefas acima descritas, incluindo os testes de validação de cada módulo, serão executadas de acordo com a seguinte calendarização:

O plano de escalonamento dos trabalhos é apresentado em seguida:

	Meses										
Tarefas	09/14	10/14	11/14	12/14	01/15	02/15	03/15	04/15	05/15	06/15	07/15
T1											
T2											
T3											
T4											
T5											
Metas	INI		M1		M2/M3					M4	M5/M6

INI Início dos trabalhos
M1 Tarefa T1 terminada
M2 Tarefa T2 terminada
M3 Tarefa T3 terminada
M4 Tarefa T4 terminada
M5 Tarefa T5 terminada

5. RESULTADOS

Os resultados dos estágios serão consubstanciados num conjunto de documentos a elaborar pelo estagiário de acordo com o seguinte plano:

M1

R1.1: Relatório de Estado da Arte

M2:

R2.1: Relatório de Definição de Requisitos.

M3:

R3.1: Relatório de Especificação

M4:

R4.1: Relatório de Desenvolvimento

M5:

R5.1: Relatório de Testes

M6:

R6.1: Relatório de Estágio

6. LOCAL DE TRABALHO

Creativity Lab Ubiwhere – IPN-Incubadora - Rua Pedro Nunes — Quinta da Nora

3030–199 Coimbra (Portugal)

7. METODOLOGIA

O aluno será enquadrado numa equipa de projecto focada na área de Machine-to-Machine, no âmbito de um projecto de I&D a decorrer na Ubiwhere, em conjunto com empresas parceiras, sendo seguida uma aproximação à metodologia ágil SCRUM, validada pela certificação da empresa em CMMI-DEV L2 e ISO 9001.

8. ORIENTAÇÃO

ISEC:

Nome (nome@isec.pt)
Categoria

Entidade de Acolhimento:
Nome (coliveira@ubiwhere.com)
Cargo: R&D Manager

9. CARACTERIZAÇÃO E REMUNERAÇÃO

- 15/09/2014
- 31/07/2014
- Horário flexível, a acordar com o aluno (8h / dia quando em full-time)
- Bolsa de apoio - 6.83€/dia de trabalho
- Integração do estagiário nas actividades correntes e possibilidade de participação no plano de formação da empresa

ANNEX B –
SMART CITIES: AN ARCHITECTURAL APPROACH

This is the paper published and presented at the International Conference on Enterprise Information Systems (ICEIS), which was held at Barcelona from the 27th to the 30th of April 2015.

SMART CITIES: AN ARCHITECTURAL APPROACH

André Duarte[1,2], Carlos Oliveira[2] and Jorge Bernardino[1,3]
[1] *Instituto Superior de Engenharia de Coimbra*
Polytechnic of Coimbra, Portugal
[2] *Ubiwhere, Lda. Coimbra, Portugal*
[3] *CISUC – Centre for Informatics and Systems of the University of Coimbra*
a21200791@isec.pt, coliveira@ubiwhere.com, jorge@isec.pt

Keywords: Smart Cities; Machine to Machine (M2M); Machine Learning; Internet of Things (IoT).

Abstract: Smart cities are usually defined as modern cities with smooth information processes, facilitation mechanisms for creativity and innovativeness, and smart and sustainable solutions promoted through service platforms. With the objective of improving citizen's quality of life and quickly and efficiently make informed decisions, authorities try to monitor all information of city systems. Smart cities provide the integration of all systems in the city via a centralized command centre, which provides a holistic view of it. As smart cities emerge, old systems already in place are trying to evolve to become smarter, although these systems have many specific needs that need to be attended. With the intent to suit the needs of specific systems the focus of this work is to gather viable information that leads to analyse and, present solutions to address their current shortcomings. In order to understand the most scalable, adaptable and interoperable architecture for the problem, existing architectures will be analysed as well as the algorithms that make them work. To this end, we propose a new architecture to smart cities.

1 INTRODUCTION

Nowadays most people live in urban areas. As populations grow, they place increasing demand on the city ecosystem and directly affect the entities responsible for the city control. These challenges make leaders adopt ways to engage with the surroundings of their city, making them more prepared and aware. The decisions they make not only directly affect the city in a short term, but are also a means to improve the decision making process. With the growth of human beings in urban areas comes a significant growth in data. This data comes from sensor networks scattered around the city or from the sensors in a smartphone. As data was produced there seemed to be a constant need to integrate all of this data to provide services, therefore, smart cities materialised.

There is a wide variety of city conceptions that have built a new horizon for cities in their challenging tasks in an increasingly cost-consciousness, competitive and environmentally oriented setting. Irrespective of whether the concept is smart city, intelligent city, sustainable city, knowledge city, creative city, innovative city, ubiquitous city, digital city or city 2.0 (e.g. Komninos 2002; Aurigi 2005; Carillo 2006; Hollands 2008, 305) they all paint a picture of a modern city with smooth information processes, facilitation mechanisms for creativity and innovativeness, and smart and sustainable service solutions and platforms (Anttiroiko et al. 2014). However, there is still a general absence of joint planning by city governments with utility providers (e.g. water, in respect of environmental sustainability) and other public services (e.g. health care). Cultural barriers include commercial confidentiality, whereas social media user groups work with open data systems, causing problems for joint working of cities with the private sector. This may create problems for collaborative ventures between city governments and businesses, and even with other public sector agencies, as well as with voluntary and community organisations. According to (Alazawi et al., 2014) a smart city depends on the provision of information, communication technologies and services to the population via web based services. However, the concept of smart city can, many times, be mistaken. In order to be smart, a city does not need state of the art technology, what it needs is interoperability between various key aspects

of the city, such as governance, finance, transportation and many others. The kind of changes that smart cities will bring to the current world are many times said to be as similar to those seen in the industrial revolution. The motivation behind the concept is the ability to improve the city ecosystem while focusing on people, allowing technology to work for them and not with them, this will result in a greater vision of society.

Furthermore this data brings many possibilities to the cities because it makes smart systems' proliferation possible. One of these cases can the smart emergency management system, which is an extremely important piece for the welfare and wellbeing of people. According to (Feng and Lee, 2010) emergency management is a dynamic and continuous process that involves preparing for disaster before it happens. If these systems are in place the probability of anticipating man-made or natural disasters increases.

The systems already in place are decentralized, which means that they do not communicate between each other, making it almost impossible to prevent disaster. This decentralization is due to the objectives of the development. Most of the times these systems are designed to address a specific case or to work as an independent system that may receive information from many parts, although without the aim to deliver information to the necessary parties.

With the intent to address these shortcomings our work will provide an architecture to a smart system in the context of smart cities. This architecture will be created with awareness of the system's possibility to scale and to adapt itself to different contexts. This architecture will address the problem of receiving the data, process it and then retrieve useful outputs to any party that subscribes to a specific type of content. This architecture can then be tuned to fit different use cases and scenarios.

The remainder of this paper is structured as follows. Section 2 presents related work on the topic and aims to cover as much information as possible; Section 3 is discusses related technologies and intends to cover technological key aspects regarding the theme; Section 4 shows functional use cases with the objective of creating a baseline to support some of the decisions made during the work; Section 5 presents an architecture for future practical application of the analysed concepts and serves to document it. Finally, section 6 presents our main conclusions and suggests future work.

2 RELATED WORK

The problem presented in this paper, has been partially developed in the past years with other studies and projects. This section provides the necessary background to understand the basis of the developed work. It is important to acknowledge that the documented analysis in the paper will be high-level, in spite of covering as much information as possible.

There are many papers that present solutions for the issue that we are working on. The rest of this section will address part of them, which we think to be the best fit for our work.

In (Vakali, Anthopoulos and Krco, 2014) the concept of smart city is discussed due to its current vagueness. Still, according to (Vakali, Anthopoulos and Krco, 2014) this concept can vary from the technologies and infrastructures of a city to an indicator that measures the education level of its inhabitants. Furthermore the work intends to analyse the SEN2SOC experiment for its impact in the current context of this topic. The SEN2SOC (SENsor to SOCial) experiment promotes interactions between sensors and social networks to enhance the quality of data in SmartSantander.

The concept of smart city is also referred and conceptualized in (Chourabi et al., 2012). The work intends to create a framework that will sketch practical implications for governments. Furthermore the work enlists some success factors for smart cities, which are: (1) management and organization; (2) technology; (3) governance; (4) policy context; (5) people and communities; (6) economy; (7) built infrastructure; (8) natural environment. The proposed framework will provide integration to all of these factors and explain correlations between them.

Although the smart city concept began to be defined in the previous work, more recent works seem to extend this concept and provide different definitions for it.

In (Piro et al., 2014) discuss that there is yet to exist a theoretical definition of Smart City, although cities are developing and shaping for not so distant future. Furthermore the work enlists some of the current definitions for the concept, that there is yet to be completely defined.

The work also enlightens the necessity of Information and Communication Technology (ICT) services, with the intent to integrate them in a generic scenario of a smart city. The approach is from a service point of view, which means that it emphasises the role of the services in the city. It is also important to refer that real world cases are shown to prove the importance of the topic.

Alongside with smart cities there are many other concepts that need to be addressed, one of them is the Internet of Things (IoT).

According to (Jara et al., 2014) this concept comprises the full ecosystem of data in smart cities, which in other words means that IoT generates massive amounts of data that need to be processed by algorithms and tools with the intent to be useful for a city. This will also provide new ways to interact with intelligent devices and create homogeneous platforms that include both machines and humans working together.

Still according to (Jara et al., 2014) this new paradigm will shape the world and create a new conception of the Internet and how people interact with it, due to the constant interconnectivity between people and the world. It will also provide the necessary resources for the creation of new applications and data driven platforms that will, hopefully, improve the citizen's quality of life.

This new way of reinventing the Internet will not only provide endless possibilities to improve the overall interaction between humans and machines but also create new challenges, which need to be tackled, to cities themselves.

Furthermore, the work aims to develop data-driven models based on human actions to act as proof of concept for Smart Cities. The system was developed using the SmartSantander testbed, which contains real-time systems and sensors scattered around the city.

Additionally the work concludes that the devices in the Internet of Things are able to gather data and provide knowledge and that a new age of interaction is about to appear, due to the increasing demand for smart applications.

In (Benkhelifa, Nouali-Taboudjemat and Moussaoui, 2014) the authors listed the current disaster management projects. The purpose of this work is to summarize existing projects regarding this matter. This work is relevant due to its diversity and detail while presenting the projects, it is extremely important to have a baseline of what was already studied and how it can, if possible, be improved. It is important to state that the focus of this work is wireless sensor networks. The most relevant outputs of this work in this context were the knowledge and awareness of the projects in this area. This listing provided a wider perspective about the topic and led to discoveries regarding the State of the Art projects, which by itself ignited the discovery of solutions and use cases for each problem.

One of the major problems encountered when dealing with large amounts of data is the system's scalability. In order to understand how similar systems operate when larger amounts of data are in place (Albtoush, Dobrescu and Ionescou, 2011) explains implementation choices that should be made in order to avoid problems. This provides useful outputs for the viability and feasibility of the system. This work also explains the necessity of risk assessment of the system, not only during the implementation but also during the working phase. Finally it is also important because it defines a framework for emergency management, which includes risk assessment and disaster prevention in a multilevel and multidimensional architecture.

With the intent of presenting the role of today's technologies in this field in (Alazawi et al., 2014) it is stated that this type of systems is growing at fast pace. In contrast to (Benkhelifa, Nouali-Taboudjemat and Moussaoui, 2014), this work focuses on Vehicular Ad hoc Networks (VANETs), sensors, social networks and Car-to-X, where X can either be infrastructures or other cars. These technologies are shaping the future with the objective of giving a ubiquitous sensing of the surroundings. Later on the work it is identified that these systems produce large quantities of data, changing the context of looking at them from small, simple solving problems, to big data problems that require stronger and more capable algorithms to be solved. Lastly it is presented a problem regarding the interoperability of these systems, which is yet to be solved. The interoperability of these systems is important due to the necessity of presenting a holistic view of the problems in the city.

In the literature there are already some papers that address the need to create a smart emergency system. A good example of this is (Radianti, Gonzalez and Granmo, 2014), where the authors present emergency systems and then start to develop a platform that intends to mimic these systems in a smarter way. The authors used a smartphone based publish-subscribe system to accomplish this. The platform helps users by sensing their surroundings and assessing the current disaster scenario, providing them with a safer way to exit the building. It is interesting to analyse the communication that was developed as it takes the data of devices and delivers it, via a web-based broker, to managers and interested parties. The broker also forwards the data to a big database where it is processed in order to retrieve sensor information in useful ways (e.g. charts, reports).

There is also another important topic to cover that is emergency management. According to (Feng and Lee, 2010) it's a process that continuously prepares for disaster even before it happens. It intends to protect people from natural or man-made disasters. It is expected that it can integrate many emergency sources to provide the best possible

outcome for the situation. The main purpose of this paper is to explore the possibility of a service-oriented architecture for emergency systems. The authors propose an architecture for this scenario and conclude that these type of systems are of extreme importance in the nowadays world.

In our work we intend to present an architecture for a generic smart system that collects, processes and delivers useful data to users. In the future a smart emergency system will be developed and will integrate information from many places, process it and then retrieve it to interested parties. It is important to understand that this work is a necessary step to accomplish a system with the minimum possible flaws. Also we will integrate technologies that lead us to a more prepared system.

3 SMART CITIES TECHNOLOGIES

Systems related with smart cities require different technologies in order to be fully addressed, therefore this section aims to cover and introduce some of them.

It is important to understand that these types of technologies are of extreme importance in this topic, some of them are directly related to the data collection and storing, while others focus on the processing part of the data lifecycle. Although this section will cover most of them, it will provide more information regarding the processing part.

To begin with, the concept of Big Data (Friess and Vermesan, 2013) shall be addressed. It is understandable that having so many information inputs (sensors, smartphones, etc.) leads to a huge amount of information that needs a new type of treatment.

In (Friess and Vermesan, 2013) the authors refer to big data as "(…) the processing and analysis of large data repositories, so disproportionately large that is impossible to treat them with the conventional tools of analytical databases." The authors also explain that this data is produced by machines, that are much faster than human beings, and according to Moore's Law this data will grow exponentially. Furthermore the authors start pointing out the major contributors for data production (e.g. web logs, RFID, sensor networks, social data, etc…).

It is also referred that Big Data requires different technologies to process the massive amounts of data within a comprehensive amount of time thus, some tools are presented in order to show the current standards in this field.

Additionally, regarding this topic, the authors explain that major companies in the big data topic have a tendency to use Hadoop (Gu and Li, 2013) due to its reliability, scalability and distributed computing.

In (Jara et al., 2014) the authors present a challenge to Big Data, which is of great relevance for our work. This challenge is, perhaps, one of the most important concepts correlated with Big Data not only because of the large amount of data but also because of the IoT paradigm.

The challenge presented is the new way of interaction between humans and the Internet via smart devices. This challenge exists, because of the way that the Internet was created, until now the Internet was based on a human to human kind of interaction, because it delivers content produced by humans for other humans. This kind of communication will not disappear, however new types of interactions will appear as smart objects integrate the nowadays world.

These new types of interactions produce large amounts of data, this is where Big Data comes into play. As has been described in this section Big Data helps us to store this large amounts of data, with the objective of being analysed by intelligent algorithms and tools to extract information and provide knowledge that will empower the applications made recurring to it.

At this point it's possible to conclude that Big Data requires special treatment as it is bigger and contains more information than typical data. For that some algorithms and tools shall be addressed with the intent to choose the most suitable to the presented system.

As posted above major companies around the world to process big data are utilizing Hadoop. Hadoop is a framework that processes big data in a distributed environment (Apache Hadoop, 2014).

Also, it is planned to scale up from single to multiple machines, where each of them provides space and computational power. This framework can also handle failures in applications. It seems like a good way to implement the system. However, in more recent works, despite being around since 2009, Spark (Gu and Li, 2013) started to be used instead of Hadoop.

In (Gu and Li, 2013) the authors made a comparison between the Spark and Hadoop aiming to show which was more suitable for production. It is important to understand that Hadoop is an implementation of the MapReduce framework developed by Google. According to (Gu and Li, 2013) this framework is not designed to support applications with iterative nature, as it cannot keep data during execution time. Because of this, at each

iteration, it needs to access disk. On the other hand, Spark, despite being a MapReduce-like framework, is designed to address its current shortcomings regarding iterative applications.

Finally the authors concluded that both frameworks are good, but their application requires a good analysis of the situation. If there is a lot of memory to run the application Spark is definitely faster than Hadoop, on the other hand Hadoop uses less memory but a lot more space in disk.

Other types of data processing are also interesting in the Internet of Things (IoT) context, due to their ability of processing data streams. For instance we can point out Complex Event Processing (CEP) (Chen et al., 2014) and Storm (Toshniwalet al., 2014). Notice that CEP is only a method of analysing and processing streams of data, on the other hand Storm is a distributed computation framework that helps with the processing of large streams of data.

CEP is defined in (Chen et al., 2014) as an effective mechanism that analyses data includes it in a context and triggers events. CEP can, for instance, analyse streams of temperature and determine if the changes in that temperature are normal or abnormal. It can also relate different types of event that lead to a single complex event, such as: (1) flames; (2) temperature spike; (3) sudden humidity decrease. From these three events the system could infer that a fire was happening. Additionally (Chen et al., 2014) aims to develop an architecture for the IoT based on distributed complex event processing. The intent behind distributed CEP is to shorten the bandwidth and the necessary computation.

Storm (Toshniwal et al., 2014) is a real-time distributed stream data processing engine that manages data streams. It was designed to be scalable, resilient, extensible, efficient and easy to administer which makes it a very robust and usable structure. Figure 1 presents a storm topology, which is the real time component that runs all the logic. Topologies are then divided in spouts and bolts. Spouts, represented by the water tap in Figure 1, and are the source of the streams of data. Bolts, represented by bolts on the topology, intend to consume the data sent by spouts, process it and then produces processed outputs.

Furthermore Figure provides a fault tolerant and scalable architecture for handling data. Additionally this architecture provides the concept of worker that can be interpreted as a node which is programmed to execute a specific task. These tasks may vary, although a good example can be using a worker to process the stream with the Esper queries. In other words each Bolt is associated with a query to be applied in the stream. This will create an efficient a quick way to process the incoming stream and query it for different types of alarming events.

Additionally this two technologies together help one another, in other words Esper needs something to organize and provide data which means that some system needs to be implemented to provide Esper with the data. This is where Storm is useful, it can handle the data management and Esper will handle the queries. This approach will join both systems to enhance both of their main capabilities when dealing with these type of data.

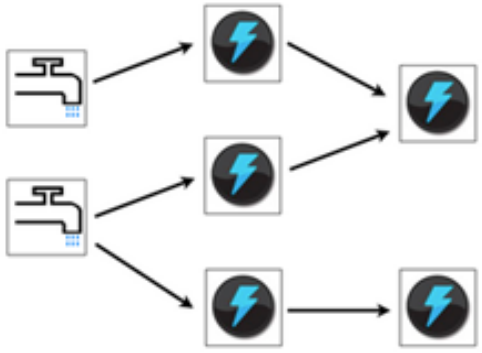

Figure 1 - Storm Topology (Apache Storm, 2014).

A very interesting aspect of Storm and CEP, is that they both can work together to provide an excellent way of processing and analysing data in our scenario.

To access this data, sensors and other devices are required. With the intent of making a more transparent communication, the concept of Machine to Machine (M2M) (Wan et al., 2012) emerged. According to (Wan et al., 2012) M2M refers to the automatic communication between, computers, sensors and other devices in the surroundings. This topic is relevant because it makes sensor-to-server communication and sensor-to-sensor possible. This allows the system to constantly check for new data and vice versa.

This concept leads us to another one related to the communication that is publish-subscribe services. According to (Ordille, Tendick and Yang, 2009) these services broadcast information to the subscribed parties. In these types of systems a subscriber is a device that will receive information from the publisher. This translates into a much more transparent system, because the publisher can send information to the subscribers and vice versa. Finally in (Radianti, Gonzalez and Granmo, 2014) their publishers are treated as the ones that generate information in the form of events. Subscribers are treated as the ones that subscribe to arbitrary flows

of information. And brokers are a middle layer between the two participants to pass along the information.

In short, these technologies, due to their relevance in this topic, seem to be an absolute need. They provide a coherent and robust ecosystem to help developers create and deploy their applications. The combination between Storm and Esper seems to be very interesting, since it provides an elegant approach to the topic.

In the latter sections some of these technologies will be addressed again, from an implementation point of view, the main goal is to provide ideas for a future implementation, leaving comments on which technology is the most suitable choice for a specific component of the architecture.

4 USE CASES

In this section current use cases of similar systems will be addressed. This will result in a better knowledge base for the current standards in the area. For this, not only examples of smart cities will be presented but also examples of emergency systems that became smarter with the inclusion of these new concepts.

Lately many smart cities have emerged, such as Amsterdam (Amsterdam Smart City, 2014), Santander (Santander Facility, 2014), Barcelona (Barcelona Open Cities Challenge, 2014), and many others. These cities, due to constant innovation projects and investments, have a tendency to be pioneers in the adoption of new standards in this field. These cities use smart systems help the decision and facilitate the decision making process.

In Finland, the city of Helsinki is running a cooperation cluster called Forum Virium Helsinki (Forum Virium Helsinki, 2014) to provide a platform to develop ICT-based services in cooperation with enterprises, public authorities and citizens as end-users. The platform is concentrated on five project areas, one of them being a smart city initiative focusing on the development of mobile phone services to facilitate urban travelling and living. It also opens up public data so that companies and citizens can create new services by combining and processing the data in innovative ways. This resembles the LivingLab movement that has spread across Europe in the 2000s (The European Network of Living Labs, 2014).

The city of Santander, for instance, uses sensors to monitor the environment, parking areas, parks, gardens and irrigation systems. These sensors are scattered around the city in order to produce alerts that will notify end users with useful knowledge of the situation.

The data is captured by an IoT node that monitors indicators such as temperature, noise or light. This data then travels through repeaters positioned in higher grounds, which send it to the gateways. Lastly this data is stored in a database or sent to other machines where it's needed.

Regarding the environmental scenario, from a user's point of view, the available indicators are the temperature, CO level, luminosity and noise, this allows them to receive useful inputs for their wellbeing throughout the day.

The environmental monitoring system is important because it shows how sensors interact with the server and how the server communicates back to the sensors and other subscribers that need this type of information. To summarise, we will discuss the "Participatory Sensing" concept (Description of implemented IoT services, 2014) to obtain a better knowledge about how users interact with the platform, and in which way is it relevant to their day-to-day life.

Figure illustrates the concept of participatory sensing from a user's point of view, which helps us understand how a typical user interacts with this kind of technologies and also how they provide useful inputs to understand the type of data a user needs during application usage. It is possible to visualise that a user can, in this case, publish events, search for events, visualise historical data, subscribe and unsubscribe to events and receive notifications.

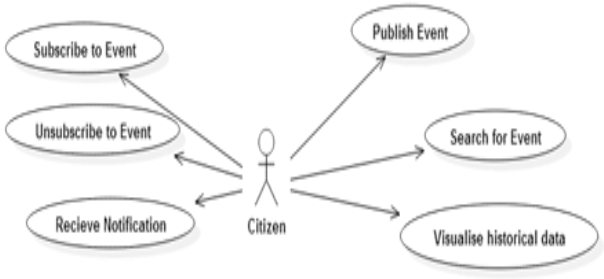

Figure 2 - Participatory Sensing - Use Case Diagram [Adapted from (Description of implemented IoT services, 2014)].

The components of the participatory sensing system are: a mobile client for end users to utilise; a server, capable of iterating through data and providing links between the apps and the SmartSantander platform also known as "Pace of The City Server"; and a module that allows devices to register onto the platform. Also, there is a system called "Universal Alert System" (UAS) system, which aims to fire user's notifications.

The "Participatory Sensing" concept allows users to actively participate in the city ecosystem. The information is then sent to the SmartSantander platform. The concept starts to get even more interesting when users become subscribers of the city systems and are able receive updates of the current status of the city or the road they have to cross to reach their destination. This type of instant real time information directly affects the city from a user's point of view due to its constant availability and usefulness. The system is available for smartphone via the app and for none smartphone users, via SMS or call.

Additionally Santander city provides other interesting case studies, which are "Precision Irrigation" and "Smart Metering".

Precision irrigation is a service that intends to provide a useful way of monitoring plants necessities and guarantee that they are fulfilled. Rather than being applied to a whole park, this system is applied by sections or individual plants. Also, the system not only focus on water management but also in other plant needs and their species and growth patterns to minimize the effort from the staff. Even though it looks a bit off the topic this system allowed to realise the necessity of designing the system to accept communications with REST and WebSockets, which are the communication technologies used by it.

Smart Metering system aims to provide IoT based solutions to monitor energy usage in offices. To address this problem new components have been added to the architecture to generate, collect and store the data and information. In addition to these, intelligent components have also been created in order to provide useful information in user-friendly way. These components provide real time analysis of data and consequent knowledge extraction. With this it can identify energy failures and reports on energy consumption that can be drilled down to a specific case.

The last system analysed was (Cecchinel, Jimenez, Mosser and Riveill, 2014), which is a prototype, named SMARTCAMPUS that aims to equip the SophiaTech campus with sensors to inspire the creation of new applications. Once more the system was chosen due to its usefulness and value in terms of possible inputs for our system.

The SMARTCAMPUS deals with many types of sensors to collect the data. To tackle this challenge the authors propose the architecture seen on Figure 3. This architecture divides in two main focal points: the message collector which intends to collect all data from the internet or sensor networks, to further store in a database that acts as a message queue; and the message processing that aims to process the messages stored in the queue. These components then store the processed information in a database.

Furthermore the architecture contains a configurator, which acts as a routine that can be called periodically to propagate a specific sensor configuration through the network. It also contains a database that contains the current sensor parameters, an API to provide an administrator interface to connect with sensors and a data API that directly accesses data to provide statistics or other types of knowledge.

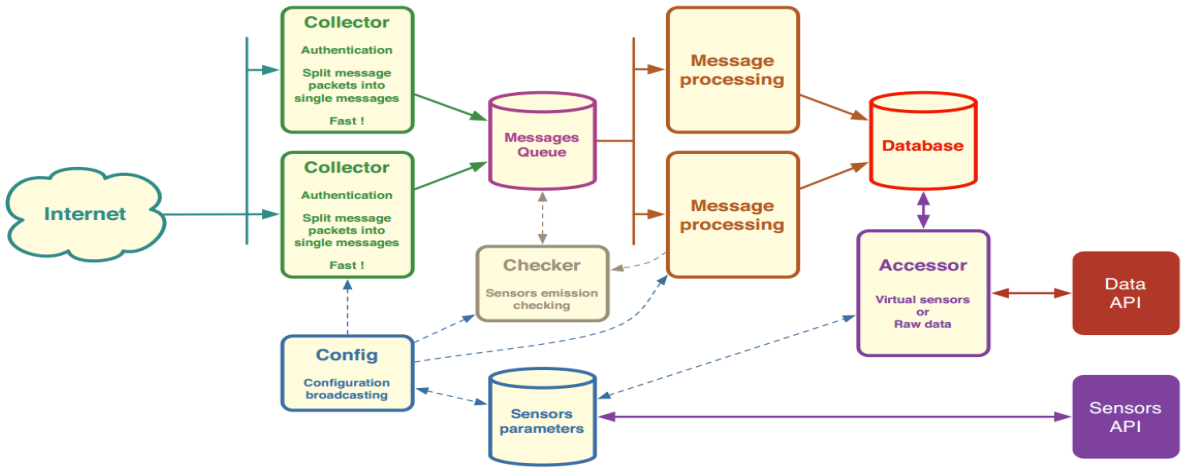

Figure 3 - Middleware Architecture (Cecchinel, Jimenez, Mosser and Riveill, 2014).

5 PROPOSED ARCHITECTURE

This section aims to present our architecture to address the typical Smart City scenario. This architecture will provide a way to gather information from many sources process it and provide useful information to the interested parties.

One of the most important things to understand is that nowadays data comes mostly in streams, which presents an issue due to the tools needed to process it. The tool that we projected to use, to process streams of data is Storm, which has already been documented in this paper. Even though Strom, by itself, cannot retrieve results one hundred percent accurate, due to being stream oriented, we plan to overcome this problem by implementing a parallel processing block with Hadoop. This will, not only provide exact results when the large amount of data is processed, but also provide a better knowledge of the data.

The approach was inspired by the lambda architecture (Lambda Architecture, 2014) with a concrete direction of using the publish/subscribe pattern. The background from other related projects allowed us to perceive that some technologies may not suit very well the collection and direct processing of data. Thus, we opted by a more complex approach that allows to a more scalable and reliable system.

This type of approach also led us to extend the capability of receiving data from multiple sources, which is extremely important in the context of IoT. Furthermore, we shall analyse the proposed architecture, present in Figure 4.

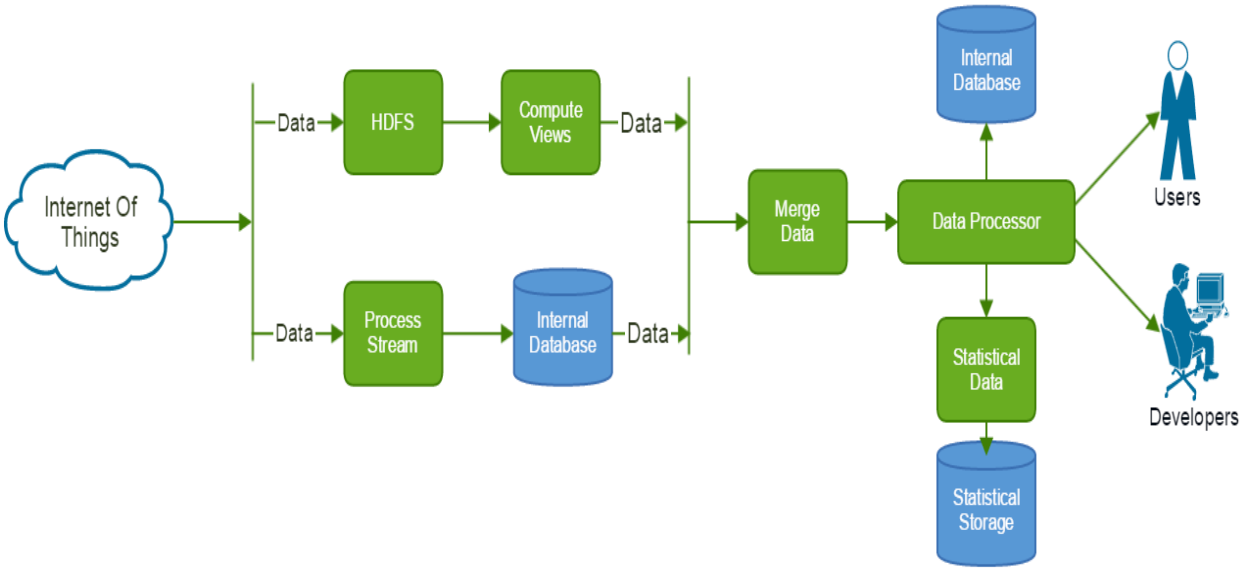

Figure 4 – Proposed Architecture.

Our architecture is projected to act as an API to provide a connection between data in the IoT and the final user, with the intent of providing relevant information regarding emergency situations.

The system will receive a data stream from IoT nodes, which is then duplicated to be processed by the batch and the speed layer. After that the data is merged with the intent of providing the result with the biggest confidence level associated. When the data is merged a bottleneck can happen, although this situation will be prevented by accepting the first result to appear with the highest confidence level. This can happen in two ways: (1) the stream layer finishes and the batch layer continues to process. With this scenario the stream layer result will be returned with a confidence level attached to it; (2) the stream and batch layer finish at the same time. In this case the data will be merged to provide the most accurate output.

After the data is merged it reaches another processing block, which intends to filter and redirect the acquired knowledge to the subscribed parties. Additionally this block sends the processed data to the statistical data block. The latter block not only keeps track of statistical data to help us understand patterns along the year but also provides data to construct KPI's, charts and reports.

After the processing is all done, users can access the data in two ways: (1) via the data API, which is projected for developers who want to build applications around this context; (2) via the data output, which will serve to return the data to the subscribed parties. Additionally the API will provide a way of notifying other sensors in the field, which means that if a sensor sends a fire alert, other sensors around it will be asked for their current situation to localize the hazard with maximum precision. This type of communication is also important if the fire is located near a road since the system can be prepared to notify street lights to prevent drivers from entering the affected road. Also in the highways a lane can be closed and the traffic redirected to other lanes or even roads.

This architecture can be applied in many different scenarios; one of them will be addressed so that we can establish an example to explain some of its functions. Let's assume we have three types of sensors: smoke, flames and temperature. These sensors are constantly sending a stream of data into our system, the idea is to process this data in order to figure out whether we are in the presence of a fire or not. The system has a threshold that serves as a maximum possible value for a normal event, when crossed they trigger events that can lead to, in this case, a fire. Having different types of sensors allows us to better understand whether the fire is happening. Different combinations of events can occur, thus the system must have something to divide the ones that are indeed problematic. Furthermore we shall materialise this example:

- If there is smoke, flames and the temperature passes the threshold, then we have a fire;
- If there is smoke, no flames and the temperature is rising, it is possible to have a fire.

Many more combinations can be presented, although these explain the concept that we are trying to achieve.

Furthermore each module of the presented architecture should be accounted for when choosing the right technologies, in order to access the full potential of it. Hence we need to account for the data stream that is arriving. For instance, it should use a publish-subscribe messaging system, which will handle the stream and split it into events that can be processed by the rest of the modules. The events that have been split will be processed by the both layers. At this point, in the speed layer, there are two important things to acknowledge: (1) it is advised to use a complex event processing system due to the nature of the system, this will provide an event based approach which will necessarily climax with event correlations and a smarter way of dealing with the data stream that constantly change. This approach will also provide the ability of integrating many types of events at once, this will expand system acceptance in terms of receiving events and inevitably prepare it to explore further sensor integrations; (2) an in memory database for storing alarming events is also useful, because of the high demand from the system.

In the batch layer algorithms with predictive capabilities should be added to enhance the system overall quality and usefulness. This will provide ways to calculate KPI's, draw charts and predict whether it is important or not to be in maximum alert level. From a high level perspective this type of inputs seem to have a great importance, with applications such as divide a specific fire protection team to a zone which is prone to peaks of fire during the summer or redirect traffic because a particular road is more likely to be affected by the floods in the winter.

The rest of the processing components in the system can be executed with any programming language and should withstand the volume and velocity of data, also the code statements should be optimized to minimize overheads and bottlenecks. The databases should be chosen according to the needs of each specific scenario. It is important to understand that many database systems can be chosen to incorporate the solution, although for each specific situation a brief

analysis of the problem should be made in order to perceive the best possible choice. As a practical example we can point out that the database in the speed layer should be in-memory, on the other hand the statistical storage could be an NO-SQL database that supports large quantities of data to enhance overall system scalability and has a good read mechanism due that its main focus is reads.

Moreover other important aspect to discuss is the communication. The way the system is designed, and from the lessons learned from the use cases, the best technologies should be REST, WebSockets and MQTT. REST will provide an easy and consistent way to access the API, providing endpoints for events and the ability to execute filters in the queries. WebSockets are useful because due to the facilitations in terms of real-time communication. The MQTT protocol is important to establish connection between the system and sensors and actuators scattered in the city in order to extract real-time information.

Additionally other important aspect is the inclusion of a message broker, which will accepts messages from the source divide the stream of data in messages that are easier to process and correlate for a better, more useful and more accurate output, which is delivered to a consumer.

6 CONCLUSIONS AND FUTURE WORK

This paper intends to document the current state of the art in smart city systems and their related technologies. Over the coming sections the problem has been documented and some use cases were studied to provide the most possible inputs with the intent to understand which challenges existed and needed to be tackled.

The analysed documents provided several useful outputs to establish a good baseline in terms of architecture and tools to be used. For instance the concept of participatory sensing, from SmartSantander, led us to think that with so much user data available the system could be adapted to process it with the intent of retrieving knowledge from it. Another example of a good tool to process data in IoT is the lambda architecture, which provides the best of the stream layer processing allied with the batch layer that provides more accurate results.

The knowledge extracted from the state of the art systems and technologies guarantees that our contributions were, as expected, scalable, adaptable, feasible and viable.

Furthermore, we aim to develop a system that will address the current shortcomings in this context. This system will be more directly related to emergency management. Therefore we aim to construct a platform that receives disaster data from many sources, process it via established components and lastly retrieves it to any party that subscribed to the specific type of event. Consequently this paper also serves as a document to establish an architecture for that type of system, serving as a first practical application of it. An initial overview of the technologies that can be used was also made with the intent of providing the necessary steps to implement a similar system, or at least provide some additional knowledge regarding this topic.

A smart emergency system is important in the current context due to its usefulness and transparency while dealing with data, as it can provide predictions and problems before they happen to managers. Thus, with the use of this type of system data becomes clearer and leads to a more prepared and quicker response to any emergency or disaster.

Another interesting application, which empowers the system, is social mining, which due to the importance of social networking in nowadays society seems like and excellent way to complement the inputs of the system.

This is important to complement the system because it can detect disasters via a post in a social network. The post does not need to be in a specific format, the algorithms will only be looking for keywords that will trigger the attention of the system. Although this data is extremely relevant, it is important to guarantee that it isn't false. A possible solution for this problem can be a request to the sensors that are placed in that specific site.

In short, Internet of Things is successfully thriving in the current world, therefore these type of systems will continue to emerge alongside it. An excellent way to evolve and prepare future cities is to be more interconnected and aware, in essence enabling better decision-making.

ACKNOLEDGMENTS

This work was partially financed by iCIS – Intelligent Computing in the Internet Services (CENTRO-07- ST24 – FEDER – 002003), Portugal.

This work was also made possible with the help of Ubiwhere, Lda, which provided useful inputs in discussions and also the facilities.

REFERENCES

Alazawi, Z., Alani, O., Abdljabar, M. B., Altowaijri, S., and Mehmood, R.. 2014. A smart disaster management system for future cities. In Proceedings of the 2014 ACM international

workshop on Wireless and mobile technologies for smart cities (WiMobCity '14).

Albtoush R., Dobrescu R., Ionescou F., (2011) A Hierarchical Model for Emergency Management Systems.

Amsterdam Smart City, (2014). [online] Available at: http://amsterdamsmartcity.com/?lang=en.

Anttiroiko, A., Valkama, P. and Bailey, S. J., (2014). Smart cities in the new service economy: building platforms for smart services. AI Soc. 29(3): 323-334

Apache Hadoop, (2014). [online] Available at: http://hadoop.apache.com.

Apache Storm, (2014). [online] Available at: http://storm.apache.com.

Aurigi, A. (2005) Making the digital city. The early shaping of urban internet space. Ashgate, Aldershot

Barcelona Open Cities Challenge, (2014). [online] Available at: http://opencities.net/barcelona.

Benkhelifa, I., Nouali-Taboudjemat, N. and Moussaoui, S. (2014). Disaster Management Projects Using Wireless Sensor Networks: An Overview. 2014 28th International Conference on Advanced Information Networking and Applications Workshops.

Carillo FJ (ed) (2006) Knowledge cities. Approaches, experiences, and perspectives. Elsevier, Amsterdam

Cecchinel, C.; Jimenez, M.; Mosser, S.; Riveill, M., (2014). An Architecture to Support the Collection of Big Data in the Internet of Things Services (SERVICES).

Chen, C., Fu, J., Sung, T., Wang, P., Jou, E. and Feng, M. (2014). Complex event processing for the Internet of Things and its applications. 2014 IEEE International Conference on Automation Science and Engineering (CASE).

Chourabi, H., Nam, T., Walker, S., Gil-Garcia, J., Mellouli, S., Nahon, K., Pardo, T. and Scholl, H. (2012). Understanding Smart Cities: An Integrative Framework. 2012 45th Hawaii International Conference on System Sciences.

Description of implemented IoT services, (2014). [online] Available at: http://smartsantander.eu/downloads/Deliverables/D4.2.pdf.

European Network of Living Labs, (2014). [online] Available at: http://www.openlivinglabs.eu/

Feng, Y. and Lee, C. (2010). Exploring Development of Service-Oriented Architecture for Next Generation Emergency Management System. 2010 IEEE 24th International Conference on Advanced Information Networking and Applications Workshops.

Forum Virium Helsinki, (2014). [online] Available at: http://forumvirium.fi/en

Friess, P. and Vermesan, O., Internet of Things: Converging Technologies for Smart Environments and Integrated Ecosystems. Aalborg, Denmark: River Publishers, 2013.

Gu, L. and Li, H., "Memory or Time: Performance Evaluation for Iterative Operation on Hadoop and Spark," High Performance Computing and Communications & 2013 IEEE International Conference on Embedded and Ubiquitous Computing (HPCC_EUC), 2013 IEEE 10th International Conference on , vol., no., pp.721,727, 13-15 Nov. 2013.

Hollands RG (2008) Will the real smart city please stand up? Intelligent, progressive or entrepreneurial? City 12(3):303–320

Jara, A.J.; Genoud, D.; Bocchi, Y., "Big Data in Smart Cities: From Poisson to Human Dynamics," Advanced Information Networking and Applications Workshops (WAINA), 2014 28th International Conference on , vol., no., pp.785,790, 13-16 May 2014
doi: 10.1109/WAINA.2014.165

Komninos N (2002) Intelligent cities. Innovation knowledge systems and digital spaces. Spon Press, London

Lambda Architecture, (2014). [online] Available at: http://lambda-architecture.net/

Ordille, J., Tendick, P. and Yang, Q. (2009). Publish-subscribe services for urgent and emergency response. Proceedings of the Fourth International ICST Conference on COMmunication System softWAre and middlewaRE - COMSWARE '09.

Piro G., Cianci I., Grieco L. A., Boggia G., and Camarda, P. 2014. Information centric services in Smart Cities. *J. Syst. Softw.* 88

Radianti, J., Gonzalez, J. and Granmo, O. (2014). Publish-subscribe smartphone sensing platform for the acute phase of a disaster: A framework for emergency management support. 2014 IEEE International Conference on Pervasive Computing and Communication Workshops (PERCOM WORKSHOPS).

Santander Facility, (2014). [online] Available at: http://www.smartsantander.eu/index.php/testbeds/item/132-santander-summary

Toshniwal A., Taneja, S., Shukla, A., Ramasamy, K., Patel, J. M., Kulkarni, S., Jackson, J., Gade, K., Fu, M., Donham, J., Bhagat, N., Mittal, S., and Ryaboy, D., Storm@twitter. In Proceedings of the 2014 ACM SIGMOD international conference on Management of data (SIGMOD '14).

Vakali, A., Anthopoulos, L. and Krco, S. (2014). Smart Cities Data Streams Integration.Proceedings of the 4th International Conference on Web Intelligence, Mining and Semantics (WIMS14) - WIMS '14.

Wan, J., Li, D., Zou, C. and Zhou, K. (2012). M2M Communications for Smart City: An Event-Based Architecture. 2012 IEEE 12th International Conference on Computer and Information Technology.

ANNEX C –
CASSANDRA FOR INTERNET OF THINGS: AN EXPERIMENTAL EVALUATION

This paper was submitted to the International Conference on Internet of Things and Big Data (IoTBD), which will be held at Rome from the 23th to the 25thof April 2016.

Cassandra for Internet of Things: an experimental evaluation

André Duarte[1], Jorge Bernardino[1, 2]
[1] *CISUC – Centre of Informatics and Systems of the University of Coimbra*
FCTUC – University of Coimbra, 3030-290 Coimbra, Portugal
[2] *ISEC – Superior Institute of Engineering of Coimbra*
Polytechnic Institute of Coimbra, 3030-190 Coimbra, Portugal
duarte.andre00@gmail.com, jorge@isec.pt

Keywords: NoSQL; SQL; Internet of Things (IoT); Cassandra.

Abstract: The proliferation of the Internet of Things (IoT) increases the amount of data that is being produced. Therefore it is extremely important to find the best possible storage engine to process these huge amounts of data. With the intent of discovering which database engine better supports the characteristics of an IoT system it is essential to analyse the existing databases and test them in a practical context. With this objective we decided to use one of the most popular databases, Cassandra, to evaluate it on an IoT environment. We evaluate the querying processing time of Cassandra using queries of an IoT real time environment and comparing it with different types of data architectures. The main focus of this work is to investigate if Cassandra can provide good performance.

1 INTRODUCTION

Nowadays the world is evolving and producing large amounts of data due to the growth of Internet of Things (IoT). This constant and fast evolution leads developers to pursuit the best possible solutions to handle large amounts of data. Even though the need for intelligent data mining tools is extremely important, we also need to pay attention to the way this data is stored and which type of engine better fits the needs of an IoT system.

To the best of our knowledge, a perfect solution for the Internet of Things data layer does not exists. With this in mind we aim to find out the best possible solution for this type of environment. Therefore an investigation was conducted to understand which database would be the most suitable to provide a production ready environment. It is important to keep in mind that, generally speaking, these kind of systems need to handle large amounts of data, real time insertion of records and huge data diversity.

The database will be used in an Internet of Things environment that needs to receive massive amounts of events in real time. This system intends to gather data from a city and process it in order to find events that are considered dangerous. The system collects data from sensors and provides alerts to each subscribed application. It is important to understand that this system will act as a demonstrator for the data layer.

In (van der Veen, van der Waaij and Meijer, 2012) it is discussed that scaling systems that deal with sensors is becoming gradually difficult due to the amount of sensors and clients that extract data from them. Therefore it is significant to not only pay attention to the frequency of the data, but also to the huge volume that it will obtain.

According to (Abramova et al., 2014a, 2014b, 2014c) Cassandra seems to have a clear advantage in terms of the characteristics necessary to implement this system because it provides good writes speeds without sacrificing performance.

Furthermore, Cassandra system was designed to run on cheap commodity hardware and handle high write throughput while not sacrificing read efficiency (Lakshman and Malik, 2010). Additionally the decision of choosing Cassandra is a result of its popularity and market share (DB-Engines Ranking, 2015). With all of this in mind, Cassandra seems to be a solid choice for this use case.

In addition to storing data, every system needs to provide it in order to query and filter later. It is

important to keep in mind that systems included in the IoT context tend to be stream oriented, rather than batch. For this reason, the database to be chosen needs to accept data in streams, or at least support a high rate of data insertion, and have the necessary mechanisms to withstand this.
We aim to test which architecture for the data layer would best suit the needs of an IoT platform in terms of querying performance, without sacrificing the write performance. There were two relevant ways in terms of implementation. First, a single table with all the data, which would then be filtered and dealt with when needed. A second approach is multiple tables for each specific application that sends events.
From a theoretical standpoint it seems that the best way of organizing our data is through the creation of a table per application. This will result in smaller tables which, in comparison to a centralized table that stores everything are a lot faster because they have much less records.
In a nutshell, we aim to understand which data architecture will have the best performance while querying the data.
The remainder of this paper is structured as follows. Section 2 gathers background on important concepts such as the IoT concept and the description of Cassandra. Section 3 describes Cassandra and its general characteristics. Section 4 defines the setup on which the tests were made. Section 5 presents the performance tests that were made. Finally, section 6 presents our main conclusions and suggests future work.

2 BACKGROUND

This section aims to present the most important concepts when referring to the general topic where the system will be implemented. On the other hand it also intends to provide the necessary background on Internet of Things and Databases.
We aim to understand which architecture is most suitable when dealing with data in an IoT environment.

2.1 Internet of Things

According to (Friess and Vermesan, 2013) the Internet of Things (IoT) "is a concept and a paradigm that considers pervasive presence in the environment of a variety of things/objects that through wireless and wired connections and unique addressing schemes are able to interact with each other and cooperate with other things/objects to create new applications/services and reach common goals."
The IoT is a concept reflecting a connected set of anyone, anything, anytime, anyplace, any service, and any network (Islam et al., 2015). The IoT is a megatrend in next-generation technologies that can impact the whole business spectrum and can be thought of as the interconnection of uniquely identifiable smart objects and devices within today's Internet infrastructure with extended benefits. Benefits typically include the advanced connectivity of these devices, systems, and services that goes beyond machine-to-machine (M2M) scenarios (Höller et al., 2014). Therefore, introducing automation is conceivable in nearly every field. The IoT provides appropriate solutions for a wide range of applications such as smart cities, traffic congestion, waste management, structural health, security, emergency services, logistics, retails, industrial control, and health care.
The Internet of Things extends even beyond communications and new services, it will allow for a future where, with everything connected, people can feel more integrated with the world and let IoT do the day-to-day recurring tasks for them.
The IoT provides a new paradigm that will shape the world and create a new conception of the Internet and how people interact with it, due to the constant interconnectivity between people and the world (Jara et al., 2014). It will also provide the necessary resources for the creation of new applications and data driven platforms that will, hopefully, improve the citizen's quality of life. This new way of reinventing the Internet will not only provide endless possibilities to improve the overall interaction between humans and machines but also create new challenges, which need to be tackled, to cities themselves.
In short, Internet of Things is successfully thriving in the current world, therefore intelligent systems will continue to emerge alongside it.

2.2 Databases

A database can be treated as a related set of information, which allows the developer to access the data via queries that intend to express his/her needs regarding that set of data in that specific timeframe, either by using simple statements or complex filtering to enhance the granularity of the search. For this data to be queried it needs to be inserted during the time that the database is in place. Nowadays there are many types of databases, however in this paper we will focus on Cassandra.

Moreover there are a lot more databases, for example we can point out some of them:

- SQL databases – these are the traditional databases that intend to store data in a structured way. Famous SQL engines are (DB-Engines Ranking, 2015): MySQL, MS SQL Server and Oracle;
- NoSQL databases – NoSQL "is used to refer to the databases that attempt to solve the problems of scalability and availability against that of atomicity or consistency" (Vaish, 2013). NoSQL databases are divided in four main groups according to each use case and architecture, these groups are:
  - Key-Value databases – These are the simplest NoSQL databases, which are based in a key-value organization that allows the developers to make CRUD (Create, Read, Update, and Delete) operations only with a key. The type of storage is BLOB (Binary Large Object) and the data structure is not organized in any fashion. According to (Redmond, Wilson and Carter, 2012) these databases have a very good performance, although aren't good for complex querying and aggregation. Examples of these databases are: Memcached (Memcached, 2015), Couchbase (Couchbase, 2015) and DynamoDB (DynamoDB, 2015);
  - Document databases – As the name implies this type of database stores and retrieves document like files, which can be XML, JSON amongst others. According to (Redmond, Wilson and Carter, 2012) a document is a hash with a unique ID which has more values related to it. Examples of these databases are: MongoDB (MongoDB, 2015) and CouchDB (CouchDB, 2015);
  - Column Family databases – These databases store data um column families which are tables with columns that are frequently accessed together. According to (Redmond, Wilson and Carter, 2012) a columnar structure "is about midway between relational and key-value". Databases of this type are: HBase (HBase, 2015) and Cassandra;
  - Graph databases – According to (Robinson, Webber and Eifrem, 2013), graph databases "are normally optimized for transactional performance, and engineered with transactional integrity and operational availability in mind". Famous databases of this type include (DB-Engines Ranking, 2015): Neo4j (Neo4j, 2015), OrientDB (OrientDB, 2015) and Titan (Titan, 2015).

In our case we opted for NoSQL databases because they seem the more appropriate fit for systems in the IoT paradigm. On the NoSQL databases we have opted for Cassandra, the next section will serve to explain our choice while introducing important topics related to Cassandra.

3 CASSANDRA DATABASE

Cassandra is a distributed storage system that manages large amounts of data across servers (Lakshman and Malik, 2010). Still according to this author Cassandra uses a combination of well-known procedures that grant scalability and availability.
In this section we will start by introducing Cassandra's data model in Section 3.1. In Section 3.2 we will explain Cassandra's data model and architecture.

3.1 Data Model

The data model of Cassandra provides a high processing speed when writing the data, this is due to the indexing.
Cassandra indexes data by key, which is a unique representation of the row that contains the data. Each row contains columns, which are attributes and finally these columns make up a column family.
Figure 1 illustrates the data model, which is composed by rows, column families and keyspaces.

Figure 1 - Cassandra's Data Model (Charsyam, 2011)

Furthermore we shall address the two important concepts that make up the data representation in Cassandra, which are the column families and the keyspaces.

- Column Family – A column family is a container for a group of rows (Hewitt, 2011). Column families are not defined, which means that the structure can be changed at any desired time, this improves the system's readiness to change and adapt during time;
- Keyspace – In Cassandra the keyspace is the equivalent to a database in the relational paradigm. The keyspace contains the column families which make up the full database. The keyspaces contain attributes that can be tuned to enhance the overall performance of the database, these attributes are:
    - Replication factor – which refers to the number of physical copies of the data. For example if the replication factor is set to two data will be replicated twice;
    - Replica placement strategy – this attribute is used for defining the strategy of how data is placed in the cluster. The possibilities to define the replicas are, Simple Strategy which is most used when we have a single group of nodes in the cluster and Network Topology Strategy which is more used when the cluster is working across multiple machines providing a way of managing the replicas in all the machines.

3.2 Architecture

In this section is given an overview of the Cassandra architecture. Cassandra uses a peer-to-peer architecture, which means that all nodes within a cluster can receive a request and respond to it (Strickland, 2014). This provides better availability when the database is online. Also, this provides redundancies which help to keep the data safe and horizontal scalability. In Figure 2 we can observe the Cassandra peer-to-peer architecture.

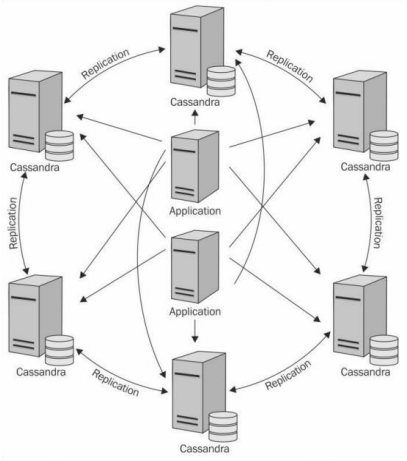

Figure 2 - Cassandra Architecture (Strickland, 2014)

Furthermore, this architecture provides high availability to the database, which means that the system does not have a large downtime period, providing constant access to the data.
In Section 3.2.1 we address the concept of replication, which allows copies of data to be stored across cluster nodes. Section 3.2.2 explains how Cassandra reads and writes the data.

3.2.1 Replication

Replication is very important in Cassandra because it provides ways of copying the data within or across nodes. This is done by storing the replicas on the keyspace they belong to. Cassandra provides two different replication strategies:

- Simple Strategy – This strategy is normally used for single data centre deployments (Datastax, 2014). When this strategy is done Cassandra will find the first replica and then will perform a clockwise movement to store the next replica. When creating this strategy the number of replicas must be defined. Figure 3 illustrates this strategy. The first replica is the original inserted value, the rest are copies placed in a clockwise fashion to replicate the data. The replication factor used was 3.

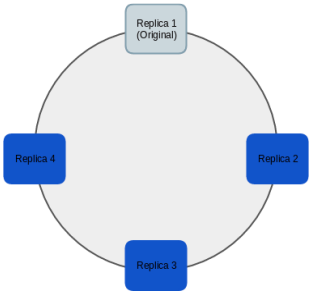

Figure 3 - Simple Strategy

- Network Topology Strategy – This strategy is used when the cluster spans across multiple data centres (Datastax, 2014). It places the replicas the same way as the Simple Strategy, although it places them in different physic groups (racks) to enhance the safety of the data in case of sudden crashes. When creating this strategy the number of replicas and the number of data centres to keeps those replicas must be defined. Figure 4 explains this strategy by providing an example. This example creates the copies in two different Data Centres with a replication factor of 3.

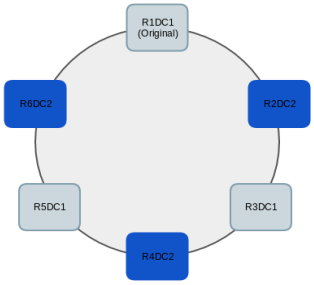

Figure 4 - Network Topology Strategy

3.2.2 Writing and Reading

Cassandra is a Column Family NoSQL database, which translates into a data format storage which is vertical oriented. The appropriateness of this database for logging systems (Abramova et al., 2014a), led us to acknowledge that it could be used in IoT.
Cassandra divides each received request into stages to enhance the capabilities while handling and serving a high number of simultaneous requests (Welsh et al., 2001). This allows Cassandra to improve its performance, however it is limited by the host machine characteristics, manly by the memory available. Finally, and because RAM memory is a lot faster than the standard HDDs and SSDs Cassandra needs to have a mechanism that will handle writing all this data to disk, in background. This mechanism is called memory mapping and consists in two similar mechanisms: the Key Cache and Row Cache. Key cache handles in memory mapping of the stored keys and it's solely responsible for storing in RAM these keys, providing fast read/write speeds on them. On the other hand, Row Cache is the memory mapping for each row (Abramova et al., 2014a).
To better demonstrate the life cycle of a record being written in Cassandra we will provide an overview of the writing architecture.
Figure 5 explains how Cassandra writes a record. First it writes every arriving row in the Commit Log, then it replicates this data on the memtable. The data is replicated in the Commit Log to ensure that there are no records lost. The data which is now on the memtable will only be written to disk when a flush happens. A flush can happen when: (1) reaches the maximum allocated memory; (2) after a specific time in memory; (3) manually by the user. When flushed the memtable becomes an immutable Sorted String Table (SSTable) which stores all the data (DZone, 2015).

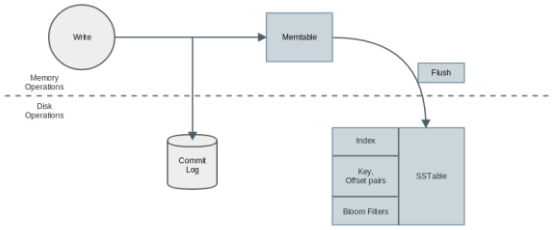

Figure 5 - Cassandra Writing

Figure 6 explains how Cassandra reads the data within one cluster. A request is made to any node in the cluster, the chosen node will become responsible for handling the requested data. The request is then processed and all the SSTables for a specific column family will be searched and the data will be gathered to merge data. Merge Data is useful because of the replication factor of the tables, for instance nothing guarantees that the data is all stored in the same table. When a read request is made it might need to gather data from multiple tables, Merge Data allows this data to be combined.

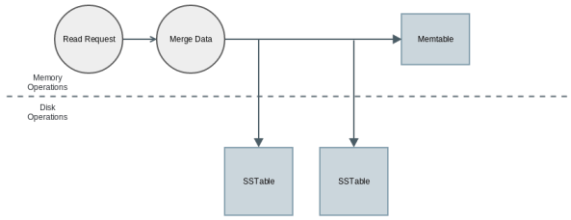

Figure 6 - Cassandra Reading

Furthermore, Cassandra provides other important features such as durability and indexing.
The durability allows Cassandra to perpetually save data, even after system crashes. Cassandra achieves this because it uses an intermediary write mechanism, which is called the commit log. Data is appended to the commit log and then saved in the database (Hewitt, 2011).
Indexing allows the queries to be faster. Cassandra stores the indexes of each column family in the node it belongs to (Abramova and Bernardino, 2013). It is also important to understand that indexing is a technique of extreme importance, especially in the IoT because it provides a way to improve the query performance.
Cassandra isn't the best storage system from a querying standpoint, although other requirements also need to be met. From the already referred approaches we quickly conclude that, from a theoretical standpoint, the most inefficient model for the data layer is the single table. This is due to the amount of data in the table which cannot be all in memory at once. In fact and from a technological standpoint Cassandra is not optimized for reading and filtering, to illustrate this we shall present an example. On a blog post from the Datastax documentation (Datastax, 2014), the author explains that when querying a table with one million records for two records, Cassandra will load the remaining 999.998 rows for nothing. This reflects in an overall performance drop while reading and filtering data.
In the next section we will explain the experimental setup which was used in the tests.

4 EXPERIMENTAL SETUP

The experiments that will be made will allow to learn which approach is better when storing data in the IoT. As mentioned in section 1 we have decided that there were two ways to organize the database which would be relevant in terms of implementation. A single table with all the data, which would then be filtered and dealt with when needed, or multiple tables for each specific application that sends events. From a theoretical standpoint it seems that the best way of organizing our data is through the creation of a table per application. This will result in smaller tables which, in comparison to a centralized table that stores everything is considerable faster because they have significantly less records. Figure 7 illustrates the two different approaches.

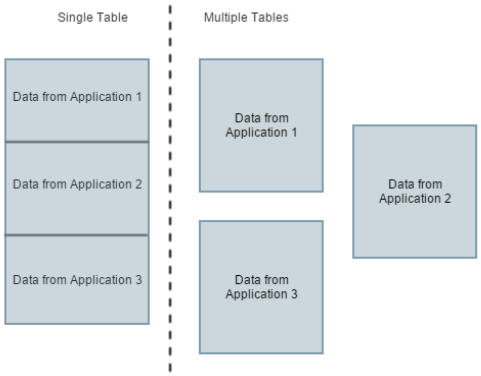

Figure 7 - Data layer possible architectures

The experimental setup was created with the following characteristics: (1) The operating system was Ubuntu 14.04 LTS 64bit; (2) The machine had a dual core, Core i5 480m with 6GB of RAM and an HDD; (3) The database ran in a single node to understand the minimum possible requirements when running the system.
We have decided not to use a benchmark tool because we have concluded that most tools available nowadays do not provide the necessary requirements to test our database system with the necessary characteristics. Also with this approach we guarantee that the performances we see are more accurate and can be replicated in a production environment.
The chosen queries intend to illustrate regular situations during the usage of the system, which reflect the better approaches to the problem, keeping in mind that attention to the write speed is also needed. To analyse them, different queries will be created, matching the needs while the system is in place. These queries may vary from time to time, although some of them will be a recurrent task that needs to be performed. Additionally, it is important to keep in mind that these queries are to be performed in an IoT system, which generates alerts with the data that comes from the sensors scattered around a city. The idea is that these alerts are filterable and searchable throughout the lifecycle of the system.

In the experiments we have the following queries:

Q1: Alert selection from a specific type – This query is performed to provide the number of alerts of each type (e.g. Number of 'warning' alerts);

Q2: Alert selection for a submitted rule – This query will be used to see how many alerts were raised by a submitted rule (e.g. how many alerts were generated by rule X);

Q3: Alert selection in a range of time – This query serves to select a type of alerts (e.g. 'warning', 'critical') in a period of time.

These queries give a broad perspective of the system in terms of querying performance.

To query the database we use the Cassandra CQL shell, to record the times we have enabled tracing which allow us to have a detailed view of the query and created indexes to allow filtering to happen.

Figure 8 - Row prototype

Figure 8 shows the row prototype which is composed by the following columns:

- alert_uuid – This field is of the type UUID, it represents the universal id of the alert to keep each alert unique;
- config_id – This field is of the type UUID, it represents the application id which created this alert;
- event_query – This field is of type TEXT and it represent the rule needed to fire the alert;
- alert_type – This field is of the type VARCHAR and represents the type of alert which was generated (i.e. Critical, Warning);
- event_type – This field is of the type VARCHAR and represents the type of event to be processed (i.e. Environment, Traffic);
- event_window – This field is of the type TEXT and represents the event window which triggered the alert;
- event_body – This field is of the type TEXT and represent the full event which triggered the alert;
- created_on – This field is of the type TIMESTAMP and it represents the timestamp on which the alert was triggered.

On the next section we will present the results of the experiments.

5 EVALUATION

In this section we evaluate the query processing time. Each chart contains, in the Y axis, the "Query Time (ms)" which represents the time the queries took to be processed. In the X axis, we have "Table Name" which represents the table where the query was made. The tables are divided by configuration and each represents an application. The "Table Name" axis uses the following notation:

- App1-App5: correspond to applications with data that comes from environmental sensors. Each of these applications have 100.000 records;
- All: corresponds to the single table containing all the information. This table will have 500.000 records.

The values presented in the experiments were obtained by executing the same query five times and then calculating the average value. Also, the first three queries of each run were discarded due to the possibility of cold boots. In the figures the dots represent the average value of the query speed and the error bars represent the standard deviation to that value.

For a better approximation of a real system, the queries were made in no specific order. This has to do with the Cassandra reading architecture which is faster if the table is in memory.

In the next sections we will show the values obtained during the experiments and present a summary of the values obtained.

5.1 Querying an alert of a specific type (Q1)

In the experiment we use this query to select all the alerts of type 'warning' from the applications. Using the CQL language the query looks like this:

SELECT * FROM query_performance.alerts _<app_id> WHERE alert_type = 'warning';

For the table with all of the data the query used was:

SELECT * FROM query_performance.alerts_full WHERE config_id = <app_id> AND alert_type = 'warning' ALLOW FILTERING;

This a very simple query, since it only lists the alerts of type "warning" that were generated by the application. However it is expected to see an enormous change in terms of performance, due to the amount of data in the "All" table. Figure 9 shows the performances for Q1.

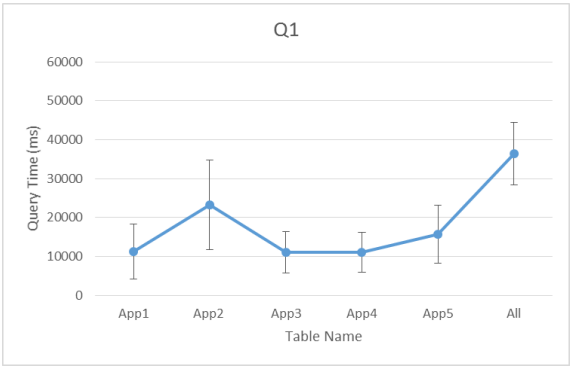

Figure 9 - Execution of Query 1

When analysing the results of Q1, shown on Figure , we can conclude that the separate tables were, in general, the best choice. Although in the second application we saw a little deviation from the average value, this is related with the reading architecture of Cassandra which is faster if the table is in memory. As explained before, we have tried to make queries to different tables in order to provide results which are useful for people who want to know if this database is a liable option for a production system.

5.2 Querying an alert for a rule (Q2)

This query intends to list every alert for a specific rule created by the user. The query, using the CQL language, will look like this:

SELECT * FROM query_performance.alerts_<config_id> WHERE event_query=<rule>;

For the full table the query looks like this:

SELECT * FROM query_performance.alerts_full WHERE config_id = <app_id> AND event_query=<rule> ALLOW FILTERING;

The query on the full table could not be completed because the operation timed out. The operation quitted when filtering the data with the where clause, this is due to the amount of data it needed to filter. We have tried to change the environment settings for Cassandra to try to overcome this situation, but the error persisted. This led to the removal of this query from the charts. Due to this problem, the comparison was made only between the applications. Furthermore, we can conclude that this query cannot be made in a production environment because the system cannot be stuck waiting for the query to end. On a real world system, and because IoT systems require near real time responses, it is impossible to implement this query because of the error it kept raising. Figure 10 shows the performances for Q2.

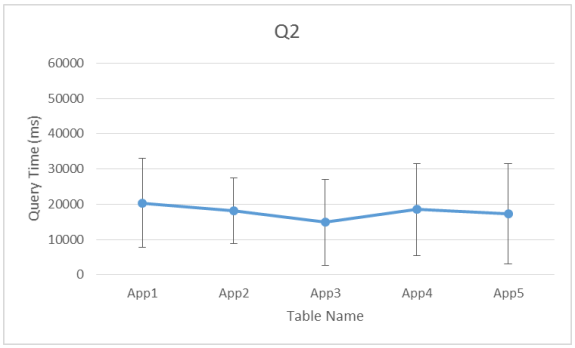

Figure 10 - Execution of Query 2

With the results of the execution of Q2, seen Figure 10, we conclude that every application has similar performances when dealing with this query. The main conclusion to draw from this experiment is that the table with all the data could not be queried because it kept raising an out of time error. This is due to the amount of data which is stored in that table which Cassandra cannot filter.

5.3 Querying an alert on a time range (Q3)

This query selects all the alerts of each application in a time range. In the real system this query is important because it delivers a time based approach to the data. Using the CQL language the query looks like this:

SELECT * FROM query_performance.alerts_<config_id> WHERE created_on <= <timestamp> AND config_id = <config_id> ALLOW FILTERING;

The query made on the table with all of the information will look like this:

SELECT * FROM query_performance.alerts_full WHERE created_on <= <timestamp> AND config_id = <config_id> ALLOW FILTERING;

Figure 11 shows the processing execution time for Q3.

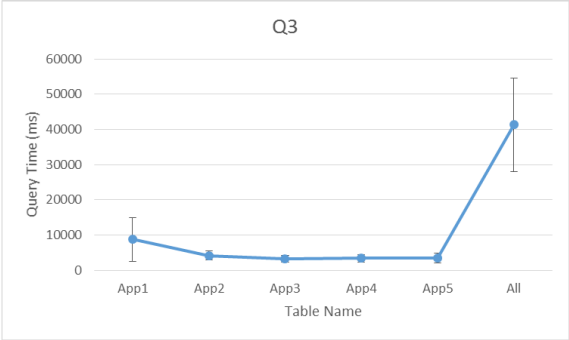

Figure 11 - Execution of Query 3

The query Q3, had comparable performance across all of the separate tables, the standard deviation on the first application is more, due to discrepancy between the performances of when the table is in memory and needs to be loaded to memory. We can also see that the average time for the table with all the data is much higher than the others, once again proving that an architecture where the data is separated is better.

5.4 Results summary

The results show that, as expected, the single table had the worst performance. This is due to the amount of data that Cassandra has to filter, which cannot be placed in memory all at once. Although the results of the "All" table were not five times worse we conclude that the best implementation is with separate tables which not only give a better performance, but also provide a better overall data separation.

The performance changes between the first two applications are a little bit different, this might be due to the size of the string that is being searched. The main differences are between the "All" table, which was finished on Q1 but not on Q2. This is due to the fact that, on these tables, data is sequentially organized which means that if the query results are not on the first records, Cassandra cannot load all the data to memory and initiate the filtering process.

The average query processing time in Q3 is a lot less than on the others, this is related to the fact that the dataset is not heterogeneous enough in terms of dates because the values of the applications were recorded on a single day. Also, filtering is made by primary key because in Cassandra to make a time range query the column with the date needs to be on the primary key of the table.

In short, we think that these queries, although very straightforward, give a quick and simple performance overview to a data layer architecture in the IoT.

6 CONCLUSIONS AND FUTURE WORK

To the best of our knowledge, a complete solution for the IoT data layer does not exist. With the intent to find a suitable and workable solution we have tested two different architectures for the data layer, which provide two different approaches when dealing with data. For this we have evaluated the NoSQL database Cassandra, which will be applied in an Internet of Things platform. The queries that were made gave us, not only an initial perspective of how Cassandra will handle the system workloads, but also will provide knowledge for whoever wants to have an idea of how Cassandra handles data in the IoT environment.

To run the tests we have tried to make constant changes to the query order to enhance the credibility of the results, this was done because the system will not have a constant pattern of querying, when deployed. This has a great impact in terms of query performance because, as we have seen before, if a table is queried twice in a row the second time it will be in memory. Additionally, it is important to refer that the tests were made on a personal computer, which makes the RAM management a lot more difficult, due to other processes that might be running at the same time.

The results show that the single table had the worst performance. From this, we conclude that the best implementation is with separate tables, which not only give a better performance, but also provide a better overall data separation. In the IoT, data is produced continuously by each application, which means that the separate tables would also be a good choice, providing an independent way for each application to store its data and be able to scale without sacrificing performance.

In summary, from this work we can conclude that Cassandra can be used on an IoT platform as the main database system because it contains the necessary characteristics to handle the overall requirements of these platforms.

The dataset used could be larger and more heterogeneous, although the results have shown differences between the two approaches. Nevertheless, tests with larger datasets and with a bigger variety of data are needed in order to understand if scalability is an issue.

As future work we suggest that similar tests can be made with sharding, which is a horizontal division of data that improves the overall performance of the queries. The main goal is to divide the applications

by shard, providing a similar approach to the separate tables we have seen. We also would like to distribute the system testing it for better availability.

REFERENCES

Abramova, V. and Bernardino, J. (2013). NoSQL databases. *Proceedings of the International C* Conference on Computer Science and Software Engineering - C3S2E '13*.

Abramova, V., Bernardino, J. and Furtado, P. (2014a). Evaluating Cassandra Scalability with YCSB. *Database and Expert Systems Applications*, pp.199-207.

Abramova, V., Bernardino, J. and Furtado, P. (2014b). Testing Cloud Benchmark Scalability with Cassandra. *2014 IEEE World Congress on Services*.

Abramova, V., Bernardino, J. and Furtado, P. (2014c). Which NoSQL Database? A Performance Overview. *Open Journal of Databases (OJDB)*, Volume 1(Issue 2).

Charsyam - Cassandra Data Model - https://charsyam.wordpress.com/tag/cassandra-data-model/ [online] Available at: [Accessed 08-01-2015]

Couchbase.com, (2015). *Couchbase*. [online] Available at: http://www.couchbase.com/ [Accessed 25 Sep. 2015].

Couchdb.apache.org, (2015). *Apache CouchDB*. [online] Available at: http://couchdb.apache.org/ [Accessed 25 Sep. 2015]

DataStax, (2014). *ALLOW FILTERING explained*. [online] Available at: http://www.datastax.com/dev/blog/allow-filtering-explained-2 [Accessed 5 Jul. 2015].

DB-Engines Ranking [online] http://db-engines.com/en/ranking (Accessed 22 April of 2015)

Docs.aws.amazon.com, (2015). *What Is Amazon DynamoDB? - Amazon DynamoDB.* [online] Available at: http://docs.aws.amazon.com/amazondynamodb/latest/developerguide/Introduction.html [Accessed 25 Sep. 2015].

Docs.datastax.com, (2015). Apache Cassandra™ 2.0. [online] Available at: http://docs.datastax.com/en/cassandra/2.0/cassandra/architecture/architectureDataDistributeReplication_c.html [Accessed 25 Oct. 2015].

DZone, (2015). *DZone Database*. [online] Available at: https://dzone.com/articles/introduction-apache-cassandras [Accessed 21 Jul. 2015].

Hbase.apache.org, (2015). *Apache HBase – Apache HBase™ Home.* [online] Available at: http://hbase.apache.org/ [Accessed 25 Sep. 2015].

Hewitt, E. (2011). *Cassandra The definitive guide*. Beijing. O'Reilly.

Höller, J.,,Tsiatsis, V., Mulligan, C., Karnouskos, S. Avesand, S. and Boyle D.,, From Machine-to-Machine to the Internet of Things: Introduction to a New Age of Intelligence. Amsterdam, The Netherlands: Elsevier, 2014.

Islam, S.M.; Kwak D., Kabir H., Hossain, M., Kyung-Sup Kwak, "The Internet of Things for Health Care: A Comprehensive Survey," in Access, IEEE , vol.3, no., pp.678-708, 2015

Jara, A.J.; Genoud, D.; Bocchi, Y., "Big Data in Smart Cities: From Poisson to Human Dynamics," Advanced Information Networking and Applications Workshops (WAINA), 2014 28th International Conference on , vol., no., pp.785,790, 13-16 May 2014

Lakshman, A. and Malik, P. (2010). Cassandra. *SIGOPS Oper. Syst. Rev.*, 44(2), p.35.

Memcached.org, (2015). *memcached - a distributed memory object caching system*. [online] Available at: http://memcached.org/ [Accessed 25 Sep. 2015].

MongoDB, (2015). *MongoDB*. [online] Available at: http://mongodb.com [Accessed 25 Sep. 2015].

Neo4j Graph Database, (2015). *Neo4j, the World's Leading Graph Database*. [online] Available at: http://neo4j.com [Accessed 25 Sep. 2015].

OrientDB Multi-Model NoSQL Database, (2015). *OrientDB - OrientDB Multi-Model NoSQL Database*. [online] Available at: http://orientdb.com/orientdb/ [Accessed 25 Sep. 2015].

P. Friess and O. Vermesan, *Internet of Things: Converging Technologies for Smart Environments and Integrated Ecosystems*. Aalborg, Denmark: River Publishers, 2013.

Redmond, E., Wilson, J. and Carter, J. (2012). Seven databases in seven weeks. Dallas, Tex.: Pragmatic Bookshelf.

Robinson, I., Webber, J. and Eifrem, E. (2013). *Graph databases*. Sebastopol, Calif.: O'Reilly Media.

Strickland, R. (2014). *Cassandra high availability*. Birmingham. Packt Publishing.

Thinkaurelius.github.io, (2015). *Titan: Distributed Graph Database*. [online] Available at: http://thinkaurelius.github.io/titan/ [Accessed 25 Sep. 2015].

Vaish, G. (2013). Getting started with NoSQL. Birmingham: Packt Publishing.

van der Veen, J.S.; van der Waaij, B.; Meijer, R.J., "Sensor Data Storage Performance: SQL or NoSQL, Physical or Virtual," Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on , vol., no., pp.431,438, 24-29 June 2012 doi: 10.1109/CLOUD.2012.18

Welsh, M., Culler, D., Brewer, E.: SEDA: an architecture for well-conditioned, scalable internet services. In: Proceedings of the Eighteenth ACM Symposium on Operating Systems Principles (SOSP 2001), pp. 230–243. ACM, New York (2001)